University of Udine
Polytechnic Department of Engineering and Architecture
Intelligent Optimization Lab (IOLab)
Ph.D. in Industrial and Information Engineering - XXXVI cycle

Ph.D. Thesis

Multi-Neighborhood Search for Combinatorial Optimization

Candidate:
Roberto Maria Rosati

Advisor:
Prof. Andrea Schaerf

Contents

Ringraziamenti

Ringrazio il mio supervisore Andrea, a cui devo la maggior parte di quello che ho imparato in questi anni. Mi ritengo fortunato ad aver avuto un mentore intelligente e perspicace, disponibile e mai avaro di tempo. Grazie per avermi lasciato sempre la libertà di sperimentare e seguire le mie idee.

Grazie a Nysret e Christian, che mi hanno accolto presso i loro laboratori, a Vienna e a Bellaterra, dandomi la possibilità di lavorare con loro e di conoscere tecniche e metodi differenti.

Grazie a Luca e Sara, colleghi e co-autori più esperti, che mi hanno accompagnato in questo percorso, e a tutti i dottorandi con cui avuto modo di condividere opinioni, gioie e dispiaceri. Una lista parziale include: Eugenia, Francesco, Francesca, Tommaso, Lucas, Felix, Florian, Tobias, Guillem, Mehmet, Camilo, Hassan, Amir, Gabriel, Pedram. Da ognuno di voi ho imparato qualcosa.

Grazie ai revisori di questa tesi, Prof. Thomas Stützle e Prof. Mauricio Resende, che l'hanno pazientemente letta e commentata, fornendomi commenti costruttivi e preziosi suggerimenti sulle possibili direzioni future di questa ricerca.

Questo traguardo non sarebbe stato possibile se non grazie al supporto incondizionato che ho sempre ricevuto dai miei genitori. Grazie a Patrizia, che ha fatto tutto quanto ha potuto per concedermi i mezzi per realizzarmi come persona. Grazie per avermi incoraggiato fin da subito alla lettura, alla scrittura e al calcolo, ricordandomi costantemente i valori dell'onestà e del rispetto. Grazie a Enzo, della cui compagnia ho goduto per pochi anni, ma che, ne sono certo, sarebbe stato orgoglioso di vedermi concludere un dottorato. Ha saputo trasmettermi la sua curiosità e la sua passione verso la scoperta. Grazie a Lino, per avermi sempre sostenuto, dandomi i mezzi e la sicurezza di avere un luogo dove tornare, per aver partecipato alle mie gioie e per il suo supporto nei momenti difficili.

Vorrei ringraziare molti membri della mia famiglia, ma per motivi di spazio non mi sarà possibile elencare né descrivere i motivi per cui ognuno di

loro ha dato un contributo per questo risultato. In una lista molto limitata, desidero menzionare i miei nonni Vincenzo e Giovanna, i miei zii Annamaria, Peppe, Nicla, Giancarlo, Giuliano, Clara, Alessandra, Annamaria. Un grazie speciale a mio zio Alfredo, per avermi stimolato fin da quando ero bambino all'uso del calcolatore e del software libero. Essere stato messo davanti al terminale di Linux in tenera età ha sicuramente avuto un'influenza nelle mie scelte future.

Un grazie particolare va rivolto a due miei amici, Eugenio e Matteo, il primo per essere stata la mia spalla nei momenti difficili e in quelli gioiosi, il secondo per avermi incoraggiato verso la strada della ricerca e per i suoi preziosissimi consigli durante questi tre anni.

Infine, grazie a mia moglie Yến, irrinunciabile compagna di vita e linfa vitale dei miei giorni, per il suo supporto a tutte le mie scelte, per avermi pazientemente accompagnato durante questi anni di dottorato, garantendomi tutta la serenità e l'amore di cui ho goduto. Poter progettare il futuro insieme è stata la mia più grande motivazione.

Acknowledgements

I thank my supervisor Andrea, to whom I owe most of what I have learned during this journey. I feel lucky to have had such an intelligent and insightful mentor, who was always generous with time and who always left me the freedom to experiment and follow my ideas.

Thanks to Nysret and Christian, who welcomed me into their laboratories, in Vienna and Bellaterra, giving me the opportunity to work with them and learn about different research aspects and methods.

Thanks to Luca and Sara, expert colleagues and co-authors, who accompanied me on this journey, and to all the doctoral students with whom I had the opportunity to share opinions, joys and sorrows. A partial list includes: Eugenia, Francesco, Francesca, Tommaso, Lucas, Felix, Florian, Tobias, Guillem, Mehmet, Camilo, Hassan, Amir, Gabriel, Pedram. From each of you I learned something.

Thanks to the reviewers of this manuscript, Prof. Thomas Stützle and Prof. Mauricio Resende, who patiently read and commented on my thesis, providing me with constructive comments and valuable suggestions on pontential future research directions.

This achievement would not have been possible if not thanks to the unconditional support I have always received from my parents. Thanks to Patrizia, who did everything she could to give me the means to realize myself as a person. Thanks for encouraging me to read, write and calculate right from the start, constantly reminding me of the values of honesty and respect. Thanks to Enzo, whose company I enjoyed for very few years, but who, I am sure, would have been proud to see me complete a doctorate. He was able to convey to me his curiosity and his passion for discovery. Thanks to Lino, for always supporting me, giving me the means and security of having a place to return to, for sharing my joys and for supporting me in difficult moments.

I would like to thank many members of my family, but for reasons of space it will not be possible for me to list or describe the reasons why each of them

contributed to this achievement. In a very limited list, I would like to mention my grandparents Vincenzo and Giovanna, and my uncles Annamaria, Peppe, Nicla, Giancarlo, Giuliano, Clara, Alessandra, Annamaria. A special thank to my uncle Alfredo, for encouraging me to use the computer and open source software since I was a child. Being put in front of the Linux terminal when I was ten years old certainly had an influence on my future choices.

A felt thank goes to two of my friends, Eugenio and Matteo, the first for being my shoulder in difficult and joyful moments, the second for encouraging me towards the path of research and for his invaluable advice during these three years.

Finally, thanks to my wife Yến, indispensable life partner and lifeblood of my days, for her support in all my choices, and for patiently accompanying me during these years of doctoral studies, guaranteeing me all the serenity and love I have enjoyed. Being able to plan the future together was my greatest motivation.

Abstract

Multi-Neighborhood Search is based on the composition of multiple local search neighborhoods. With respect to a single neighborhood, it provides a better connectivity in the search space and gives access to different search patterns, while also reducing the risk of getting stuck in a particular region of the search space.

In this thesis, we present a methodological approach to design Multi-Neighborhood Search methods under a stochastic framework. Neighborhoods are balanced through fixed rates and internal biases. We also investigate the use of reinforcement learning for the adaptive tuning of neighborhood rates. We discuss the interaction with the search space, the objective function and the metaheuristic that guides the search.

We validate our approach on four challenging combinatorial problems spanning various domains: the Minimum Interefence Frequency Assignment Problem, the Sports Timetabling Problem, the Healthcare Routing and Scheduling Problem, and the Capacitated Dispersion Problem. To solve them, we design Multi-Neighborhood Search methods that utilize from three to six distinct neighborhoods each, some of them originally conceived. We use Simulated Annealing as the metaheuristic that guides the search.

We compare our results on instances from benchmark datasets and we show that Multi-Neighborhood Search outperforms quite consistently the state-of-the-art methods from the literature. Furthermore, we demonstrate through experimental results on the Sports Timetabling Problem and the Examination Timetabling Problem that the method's robustness can be enhanced with reinforcement learning.

Finally, we replicate the schemes from Multi-Neighborhood Search to the matheuristic Construct, Merge, Solve and Adapt (CMSA), including the use of reinforcement learning, to shape the Multi-Constructor CMSA, that we apply to the Maximum Disjoint Dominating Sets Problem. Moreover, we also employ CMSA to solve a real-world Bus Driver Scheduling Problem with complex break constraints.

Chapter 1

Introduction

Local search is a procedure based on the idea of beginning from an initial problem solution and iteratively making small adjustments to progress toward better solutions. It closely mirrors the way we, as humans, approach practical problems. Consider a scenario where we've recently moved to a new house and need to organize our belongings on shelves in different rooms. Like most people, we would initially arrange our items based on an idea in mind or a plan predetermined beforehand. However, after considerable physical effort, we might realize that the new house doesn't look exactly like we were dreaming, and improvements can be made. At this point, it is rare for someone to decide to remove all the items they've arranged over many hours of work and start again from scratch. Instead, we typically refine the current state by iteratively adjusting the positions of individual items until we achieve an overall satisfactory layout. This process closely resembles how local search works in the context of combinatorial optimization. Naturally, moving single objects around might not be enough to efficiently reach the desired state. In addition, we can swap the positions of pairs of items, remove items no longer needed, insert items that we can buy from the shop across the street or that we have previously removed, and perform even more complex moves, like a permutation of books in a bookcase that changes their order from alphabetical to chronological. Multi-Neighborhood Search operates on this principle, by leveraging the combination of multiple local search neighborhoods rather than relying on a single one.

Our research is based on the hypothesis that a well-chosen combination of local search neighborhoods within a *multi-neighborhood* structure, stochastically explored and exploited by a well-established metaheuristic like Simulated Annealing (SA), can significantly enhance solution capabilities

in tackling challenging combinatorial problems. Therefore, the primary focus of this thesis is Multi-Neighborhood Search (MNS). Beyond this core focus, we also explore the transference of MNS principles to the matheuristic Construct, Merge, Solve & Adapt (CMSA), introducing the novel concept of Multi-Constructor CMSA. Finally, we investigate the use of a common reinforcement learning strategy to automate the choice of the policy for operator selection in both MNS and Multi-Constructor CMSA.

To validate our hypothesis, we implement and test our solution methods on a diverse set of real-world combinatorial problems: Minimum Interference Frequency Assignment, Sports Timetabling, Home Healthcare Routing and Scheduling and Capacitated Dispersion for what concerns MNS, and Bus Driver Scheduling and Maximum Disjoint Dominating Sets for what concerns CMSA. They are characterized by their complexity, rich sets of constraints, and large-scale instances.

We assess the contributions of individual components through statistically-principled analysis and compare the results against state-of-the-art algorithms on instances from benchmark datasets from the literature.

In the following, we introduce key concepts in design of algorithms for combinatorial optimization, discuss the motivations for this research, and provide an overview of the thesis structure and content.

1.1 Context

When we talk about a combinatorial problem, we assume that it contains, in part or completely, discrete variables, that can only assume values from a finite set. Solving an instance of a combinatorial problem consists in selecting a solution S in the solution space $\mathcal{S}$, which comprises all the possible solutions to the problem instance. We distinguish into *decision problems*, that are solved answering a question about whether a solution exists or not, *search problems*, that, in addition, require to determine the actual solution, and *optimization problems*, that are solved by finding a solution that minimizes or maximizes a given function $f : \mathcal{S} \to \mathbb{R}$, called objective function. If the problem is a minimization one, the goal is to find a solution $\bar{S}$ such that $f(\bar{S}) \leq f(S), \forall S \in \mathcal{S}$. Conversely, if the problem is a maximization one, the goal is to find a solution $\bar{S}$ such that $f(\bar{S}) \geq f(S), \forall S \in \mathcal{S}$. The solution $\bar{S}$ is known as *global minimum* (respectively, maximum), or *global optimum*. In this thesis, we focus on combinatorial *optimization* problems.

In principle, to solve to the optimality a combinatorial optimization

problem, we could enumerate all the combinations of discrete values that the variables can assume and select, among them, the combination associated with a feasible solution of minimum cost. For example, if we have 10 binary variables, we can easily enumerate the $2^{10} = 1024$ solutions. However, this number grows exponentially with the number of the input variables, and with only 100 binary variables (which would be a small instance in most problem) the number of combinations is already $2^{100} = 1267650600000000060967983513600$. Enumeration can be made more efficient in many ways, for example by using a backtracking technique, that prunes branches that surely leads to infeasible or not optimal leaves. Even so, however, the number of combinations is likely to remain too large to be computed in a reasonable amount of time. Therefore, in most real-world cases, and in all the problems discussed in this thesis, we need to resort to more complex mathematical or algorithmic methods.

Solution methods for combinatorial optimization are divided into exact methods, that guarantee that a proven optimal solution $\bar{S}$ is found (including the aforementioned enumeration technique), and heuristic methods, that do not provide such guarantee. The challenge in solving combinatorial problems comes from the discreteness of the variables. Indeed, if we only had continuous variables and linear constraints, we could formulate a linear programming model of the problem, that can be solved in polynomial time (Khachiyan, 1979), for example by means of the interior-point method[1] (Karmarkar, 1984). However, the discreteness of the variables makes problems much harder. Besides discreteness, many combinatorial problems are also nonlinear or nonconvex and are known for being NP-complete or NP-hard, meaning that no polynomial-time algorithm exists for their solution[2] (Karp, 1972). There are, naturally, exact methods that can solve to the optimum combinatorial problems. A majority of them are based on or derived from the Branch and Bound procedure, originally proposed by Land and Doig (1960). In specific cases, exact algorithms have obtained remarkable results, like in the case of the Traveling Salesman Problem (TSP, Applegate et al., 2009). However, the computational complexity of combinatorial problems is such that, even with the most powerful computers that we have nowadays and we will have in the next decades, in the large majority of situations it is not and it will not be possible to design exact algorithms that solve them to optimality in reasonable time. This consideration extends to quantum computing: while there is no

[1] The well-known simplex method is not polynomial in the worst-case scenario.
[2] Unless $P = NP$

consensus yet on the achievements that quantum computers will deliver for combinatorial optimization, at the moment they cannot provide more than a quadratic speed-up with respect to the classical computer on NP-complete problems (Bennett et al., 1997; Preskill, 2023), which is achieved by the Grover's algorithm (Grover, 1996). Therefore, the design and ideation of heuristic algorithms will remain for long time the most viable method to tackle combinatorial problems.

Under the name of *heuristics*, many different algorithms are grouped, from simple greedy algorithms to more complex methods. They renounce to the guarantee of finding the optimal solution but exploit the knowledge of the problem and of the solution space in order to reach good solutions in a limited number of iterations. As a well-established algorithmic paradigm, local search is at the basis of many successful heuristics, such as the Lin-Kernighan heuristic for solving the TSP (see Lin and Kernighan, 1973; Helsgaun, 2000).

While a heuristic is tailored on the problem it was designed for, a *metaheuristic* is a problem-independent framework for the development of problem-specific heuristics (*meta* comes from the Ancient Greek word μετά, that means "after", "beyond"). This term seems to have been suggested by Glover (1986). Historically speaking, metaheuristics gained popularity in the 80s. Famous metaheuristics based on local search are Simulated Annealing (SA, Kirkpatrick et al., 1983; Černý, 1985), Tabu Search (TS, Glover, 1989, 1990), Greedy Randomized Adaptive Search Procedures (GRASP, Feo and Resende, 1995), Variable Neighborhood Search (VNS, Mladenović and Hansen, 1997), and Iterated Local Search (ILS, Lourenço et al., 2003).

Not all metaheuristics are based on local search. Other paradigms encompass *population-based* metaheuristics, that evolve a population of solutions instead of a single one, *matheuristics*, that partially destroy solutions and repair them with an exact solver, and *hyper-heuristics*, where the operators are low-level heuristics. Among population-based methods we mention Genetic Algorithms (GA, Holland, 1992), Particle Swarm Optimization (PSO, Kennedy and Eberhart, 1995), and Ant Colony Optimization (ACO, Dorigo et al., 2006). About matheuristics, we remark about Large Neighborhood Search (LNS), proposed by Shaw (1998) and made famous in its adaptive variant by Ropke and Pisinger (2006), Local Branching (Fischetti and Lodi, 2003) and CMSA (Blum et al., 2016). For what concerns hyper-heuristics, we forward the reader to Burke et al. (2003, 2013, 2019).

A more recent trend in combinatorial optimization, in general, and in metaheuristics, specifically, is the use of machine learning inside the

algorithm to simulate expert behavior or to learn the best policy for certain algorithmic decisions (Bengio et al., 2021).

The idea of systematically studying the different compositions of local search neighborhoods was proposed by Di Gaspero and Schaerf (2006). Even though the possibility of using multiple neighborhoods, mostly in a sequential manner, had already emerged in metaheuristics like VNS and ILS, the novelty in the approach of Di Gaspero and Schaerf was the idea of considering the multi-neighborhood as an independent algorithm component from the metaheuristic. To date, we find in the literature a discrete number of successful applications of the Multi-Neighborhood Search paradigm, to solve a variated set of problems, including Examination Timetabling (Di Gaspero and Schaerf, 2003; Bellio et al., 2021), the Traveling Tournament Problem (Anagnostopoulos et al., 2006; Ribeiro and Urrutia, 2007; Di Gaspero and Schaerf, 2007), the Maximum Weight Clique Problem (Wu et al., 2012), Patient Assignment in healthcare (Demeester et al., 2010; Ceschia and Schaerf, 2011), and Vehicle Routing (Ceschia et al., 2011; Jin et al., 2012; Soto et al., 2017). In the large majority of cases, the underlying metaheuristic is TS or SA.

1.2 Motivations and objectives

Earlier studies have shown the benefits of Multi-Neighborhood Search, that enhances connectivity in the search space, offers access to diverse search trajectories, and reduces the likelihood of being trapped within a specific region of the search space. However, despite the substantial growth in the research interest towards metaheuristics, the combination of multiple neighborhoods, especially in the context of stochastic exploration, has not been systematically investigated after the work of Di Gaspero and Schaerf.

This gap in the literature has been noticed by Franzin and Stützle (2019). After analyzing in their article "Revisiting simulated annealing: a component-based analysis" all the variants of SA emerged over almost four decades of research, the authors conclude that the key components for the algorithm performance are the acceptance criterion and the neighborhood exploration. Not only they derive this clear result from data, but they literally comment: "More *surprising* is maybe the high importance for the neighborhood exploration, a component *often neglected* in the SA literature".

The lack of interest for the neighborhood relation, that happened in spite of the fact that it is indisputably the core of the whole local search paradigm, might be due to various reasons. One reason is that it is often

taken for granted in metaheuristic design and, as pointed out by Franzin and Stützle. Another factor is the trend towards the continuous development of new metaheuristics that has occurred over the last two decades, often resulting in the disregard of basic algorithmic components in the pursuit of constant novelty.

In fact, boosted by the initial success of methods inspired by biological phenomena, such as the aforementioned GA, PSO and ACO, we have assisted to a dramatic increment in the number of published articles introducing supposedly new metaheuristics that exploit metaphors of natural, biological, and anthropological phenomena (which we, on purpose, do not mention here). This trend has nonetheless drawn the attention of some meticulous scholars who have shed light on the fact that many of them are just disguises of well-known methods under new catchy names and have contributed very little, if anything, scientifically (see Weyland, 2015; Sörensen, 2015; Aranha et al., 2022; Camacho-Villalón et al., 2023). Since then, a significant portion of the scientific community has become increasingly critical of this worrisome way of conducting research, that obfuscates actual improvements and creates confusion about the various methodologies, especially in researchers and practitioners from outside the field. Moreover, it drains energy away from the actual focus of our research, that is the improvement in our capabilities to solve hard combinatorial problems, gaining comprehension of what are the components that make algorithms effective.

With the recognition of the importance of placing neighborhood design back at the center of the research agenda, this thesis aims at establishing a methodology for the design of competitive stochastic Multi-Neighborhood Search algorithms. Our approach embraces how neighborhoods are composed together, as well as their behavior and interaction with each other and with the search space, their relation with the cost function, and their integration inside a metaheuristic. In addition to developing a structured approach, we design and implement actual Multi-Neighborhood Search solution methods capable of challenging the current state-of-the-art across a diverse range of combinatorial problems. Furthermore, we foster the transition from the static nature of the neighborhood rates toward an adaptive approach that makes use of reinforcement learning. Finally, we show that it is possible to transfer certain principles derived from Multi-Neighborhood Search to another method, the matheuristic Construct, Merge, Solve & Adapt (CMSA), through the introduction of the concept of Multi-Constructor CMSA.

Aware of the importance to step back from the metaphor trap, in conducting our research we join the choir of scientists advocating for

simplicity and clearness in the algorithm design approach and in its disclosure. We believe that the proper way of conducting our research is through a rigorous methodology based on statistically-principled analysis of the contribution of individual algorithm components. This methodology aims to retain only useful components and at to keep algorithm design as simple as possible, while avoiding vague and complicated metaphors that divert from the actual essence of the algorithm.

1.3 Structure of the thesis

This thesis is organized into four distinct parts, each dedicated to a specific aspect of the research.

Part I is titled "Methods" and introduces and elaborates on the fundamental techniques employed in the study. It includes chapters Chapters 2 and 3 on Multi-Neighborhood Search and CMSA.

Part II is titled "Applications of Multi-Neighborhood Search" and delves into the design of Multi-Neighborhood Search methods to solve real-world problems. It covers Chapters 4 to 7, about the Minimum Interference Frequency Assignment (MIFAP), the Sports Timetabling Problem (STT), the Home Healthcare Routing and Scheduling Problem (HHCRSP) and the Capacitated Dispersion Problem (CDP).

Part III is titled "Applications of CMSA" and covers Chapters 8 and 9. It includes a practical application of the classic CMSA to a Bus Driver Scheduling (BDS) problem, and the introduction of the Multi-Constructor CMSA, which is applied to the Maximum Disjoint Dominating Sets Problem (MDDSP).

The final part, Part IV, titled simply "Reinforcement Learning", explores the integration of reinforcement learning techniques into both Multi-Neighborhood Search and Multi-Constructor CMSA.

The manuscript is completed by the conclusions, at Chapter 11.

This thesis presents work that has been published or submitted for publication in peer-reviewed journals and conferences. With reference to the list of publications, we report in Table 1.1 the correspondence between the chapters and the publications. Other publicly available material include the source code, instance generators and validators, instance solutions and new generated datasets. When present, they are indicated in the respective chapters.

Table 1.1: Correspondence between chapters and publications.

Chapter	Topic	Publications	Reference
Chapter 4	MIFAP	1	Ceschia et al. (2022)
Chapter 5	STT	2,8	Rosati et al. (2022); Van Bulck et al. (2023)
Chapter 6	HHCRSP	9	Ceschia et al. (2024a)
Chapter 7	CDP	10	Rosati and Schaerf (2024)
Chapter 8	BDS	6	Rosati et al. (2023b)
Chapter 9	MDDSP	3,5	Rosati et al. (2023a, 2024)
Chapter 10	RL	7	Ceschia et al. (2024b)

Part I

Methods

Chapter 2

Multi-Neighborhood Search

The local search paradigm, also known as neighborhood search, is based on the idea of starting the exploration of the search space from a solution of the problem and then moving, at each iteration, to a neighboring solution. This is done by applying small perturbations to the values of one or more solution components.

In this chapter, after a recall of the general notions of local search, we describe in detail our Multi-Neighborhood Search approach. We assume, without loss of generality, that we deal with a minimization problem.

2.1 Local search concepts

The *neighborhood relation* $\mathcal{N}$, also know simply as *neighborhood*, defines the transformation rule that is employed to move from the current solution to a neighboring solution. A *neighbor* is a solution that can be reached from the current candidate solution through the application of the neighborhood relation $\mathcal{N}$. For any given solution S, $\mathcal{N}(S)$ is the set of neighbors.

The *size* of a neighborhood $\mathcal{N}$ with respect to a solution S is its cardinality $|\mathcal{N}(S)|$. If, on average, $|\mathcal{N}(S)|$ is small, we say that a neighborhood is *small*, otherwise we talk of a *large* neighborhood.

A *move* is a concrete application of the neighborhood rule that transforms a solution S_i in a solution $S_j \in \mathcal{N}(S_i)$, and we denote it with the letter m. A move modifies selected *solution components*. We say that move $m\langle s, v\rangle$ assigns value v to component s. The notion of solution component depends on the specific problem representation, but, to generalize, we can think of them as the variables of the problem. Neighborhoods that only modify one or a limited number of components are called *atomic*, and are typically small,

while neighborhoods that modify a great number of components are typically large.

The *cost of a move* is the difference between the cost of the new solution and the cost of the previous solution. If a move brings from S_i to S_j, the cost of the move is defined as $\Delta f(S_j, S_i) = f(S_j) - f(S_i)$. It is also called *delta cost* or *differential cost*. If $\Delta f(S_j, S_i) < 0$, the move is called *improving* move, if $\Delta f(S_j, S_i) > 0$ it is called *worsening* move, if $\Delta f(S_j, S_i) = 0$, the move is called *sideways* move.

A *local minimum* for the neighborhood $\mathcal{N}$ is a solution S^* such that $f(S^*) \leq f(S)$, $\forall S \in \mathcal{N}(S^*)$, that is, no improving moves can be found among the neighbors of solution S^*. A *strict local minimum* is a solution S^* such that $f(S^*) < f(S)$, $\forall S \in \mathcal{N}(S^*)$, that is, in addition to improving moves, also sideways moves are not found among the neighbors of solution S^*. A local minimum is also known as *local optimum*. Differently from the concept of global minimum, that is a property of the instance, the local minimum depends on the neighborhood. A local minimum for the neighborhood relation $\mathcal{N}_i$ is not necessarily a local minimum for a different neighborhood relation $\mathcal{N}_j$.

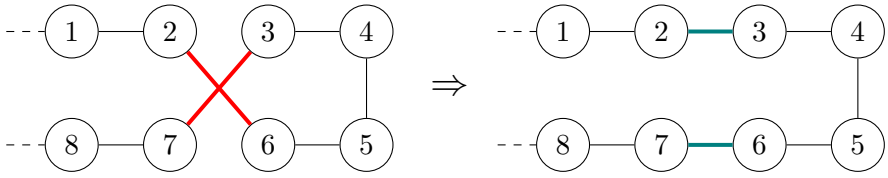


Figure 2.1: Execution of a 2-opt move for the TSP.

Fig. 2.1 shows an example of local search neighborhood. It is the 2-opt neighborhood for the TSP (Flood, 1956; Croes, 1958), that takes two edges in the route and exchanges them with two edges not in the route. On the left, we have the solution before the move, and on the right the solution after its application. For simplicity, we only draw the part of the route affected by the move.

2.1.1 Search space

The *search space* Σ is the set of all the valid solutions that can be accessed by the search algorithm. We say a solution is *valid* if it is allowed in the search space, independently from its feasibility.

The search space can be restricted to make the search more efficient. The restriction is done by imposing additional rules on the validity of the solution,

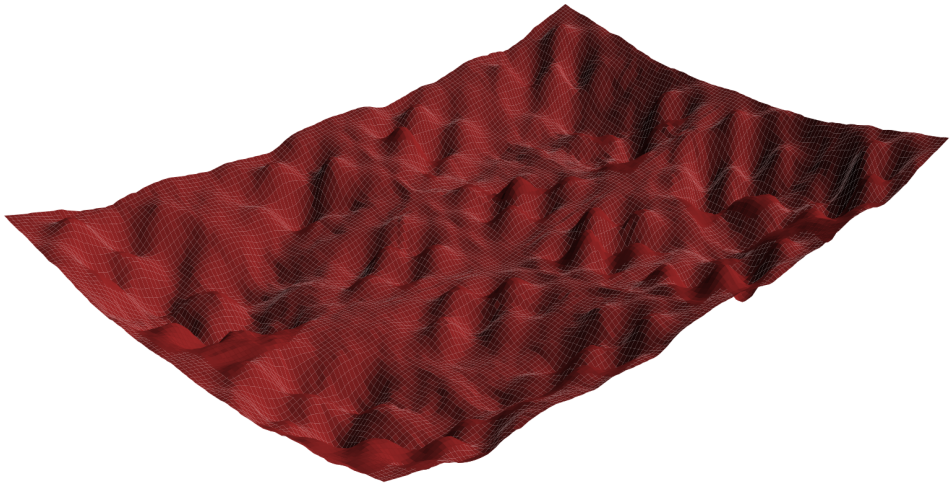


Figure 2.2: Search spaces are complicated landscapes, affected by ruggedness, plateaus, basins of attraction and several local minima

that preclude the possibility to reach certain feasible solutions. Conversely, one may consider expanding the search space to explore the infeasible regions. It is helpful especially when promising regions in the search space are interconnected solely through paths that traverse the infeasible region. However, it is essential to acknowledge that including the infeasible region significantly enlarges the search space. Often, a balance is found by allowing the violation of certain hard constraints while excluding others from the search space.

The search space of a combinatorial problem is a complex landscape characterized by *ruggedness*, *basins of attractions*, and *plateaus*. The *ruggedness* is higher where there is a greater density of local minima, and, in general, if the search landscape is not smooth. The search can be particularly hampered if these local minima are shallow, as it might get repeatedly trapped in them and miss the good descents. The *basin of attraction* of a local minimum is the region in which the search is "attracted" by the local minimum. Basins of attraction might be mistaken for promising descents. They can trap the search in a region of the search space where the objective function is not generally good, and lead to idling around the same local minimum. A *plateau* is a region where the objective function is constant. The presence of plateaus is problematic for local search, because they make it difficult to find the direction that lead to better solutions. A visual representation of a search landscape is provided in Fig. 2.2, where the

elevation represent the objective function value.

2.1.2 Cost function

The *cost function*, denoted as $F(S)$, is the internal function used by the local search algorithm to evaluate the cost of the moves. In some cases it coincides with the objective function $f(S)$, but, in general, it can differ. This typically happens when we include the infeasible region in the search space. In that case, the cost function is designed to penalize the violation of the hard constraints, in addition to minimizing the objective function, as follows:

$$F(S) = w_h H(S) + f(S) \tag{2.1}$$

where $H(S)$ is a measure of the violations of the constraints and w_h is a suitably high weight that penalizes the violation of the constraints. A higher weight given to w_h makes solutions in the infeasible region less desirable, while a lower weight makes the infeasible region more attractive.

In problems that present many plateaus, we can employ a *lexicographic cost function*. Taking advantage of expert knowledge, we add an artificial objective that discriminates among solutions in the plateau, and guide the search toward more promising areas in the search space. To keep the cost function linear, the new cost component can be combined inside $F(S)$ with a very low weight, so that it is taken into account only when the other objectives are equal.

2.1.3 Neighborhood exploration and acceptance criterion

In local search, the *exploration criterion* defines how the neighborhood is explored, while the *acceptance criterion* defines whether a neighbor is accepted as new current solution.

The exploration criterion can be sequential or stochastic. Given a solution S, the *sequential exploration* starts from a neighbor $S_i \in \mathcal{N}(S)$ and then explores sequentially the other neighbors. Therefore, to perform a sequential exploration of the neighborhood, a rule that allows a ordering of the moves is needed. In case we evaluate all neighbors from the first to the last, we talk about exhaustive exploration. Naturally, exhaustive exploration is computationally expensive, especially for large neighborhoods. The alternative is *stochastic exploration*: a neighbor is selected at random, usually according to the uniform distribution.

The most used acceptance criterions are: *first improving*, *best improving*, and *cost difference*. First improving criterion accepts the first improving

neighbor that is encountered, either sequentially or by repeated random draws. Best improving accepts the best improving among all the neighbors, and it is coupled with sequential exhaustive exploration. Cost difference imposes accepting a neighbor if the difference of cost between the neighbor and the current solution is below a certain threshold.

It is not uncommon to see in the literature that the notions of neighborhood, the exploration rule and the acceptance criterion are confused. For example, the actions "choose the best improving move in $\mathcal{N}(S)$" and "choose a random move in $\mathcal{N}(S)$", refer to different exploration and acceptance criterion, but are sometimes regarded as distinct neighborhoods. In this thesis, we make a clear distinction between these concepts.

2.1.4 Local search and metaheuristics

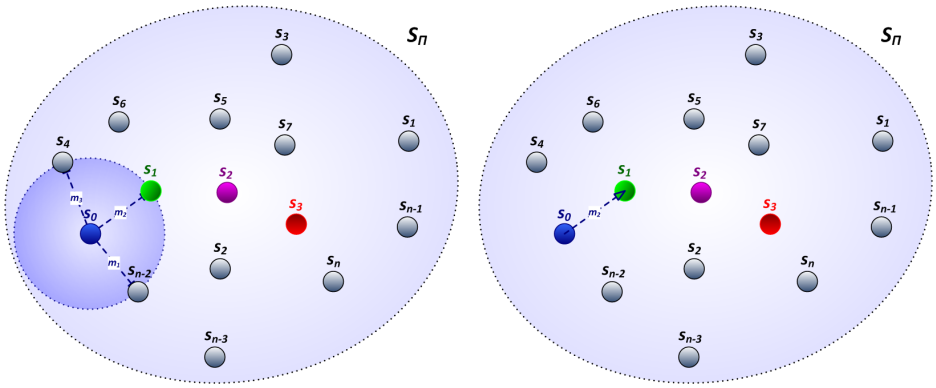
The term local search itself does not refer to a specific algorithm, but rather to an optimization paradigm based on the neighborhood relation. Therefore, to execute local search, we have to design a local search algorithm.

Fig. 2.3 shows an example of execution of a local search algorithm. It starts from an initial solution and, iteratively, explores the neighboring solution and moves to one of the neighbors, until the stop criterion is met.

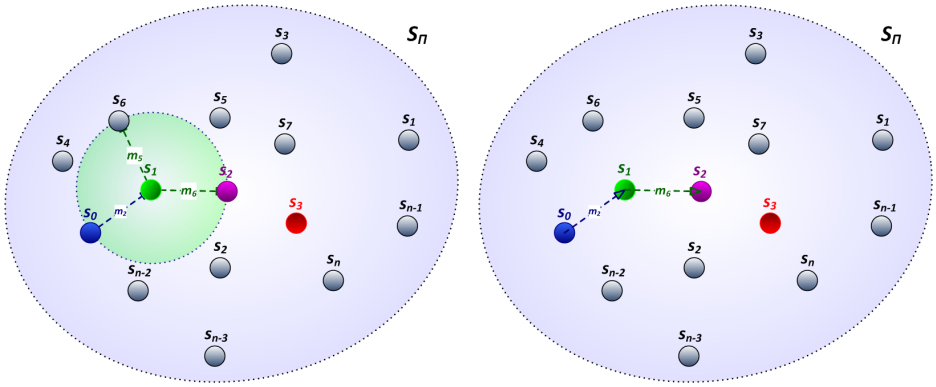
The simplest local search algorithm is the following: first, determine at random or with a constructive heuristic an initial solution $S_i \in \mathcal{S}$. Then, find the best improving neighbor $S_j \in \mathcal{N}(S_i)$, that is, $S_j : \Delta F(S_j, S_i) \leq \Delta F(S_j, S), \forall S \neq S_j \in \mathcal{N}(S_i)$, and move to it. Third, repeat iteratively the step until a local minimum is found and no improving moves are possible. This algorithm is also known as *steepest descent*. It is completely unbalanced toward exploitation, efficiently finding a local minimum but offering limited opportunities for discovering good local minima.

It is important to acknowledge that the effectiveness of a local search algorithm is determined by its ability to escape local minima, basins of attraction, and navigate plateaus, rather than just the way it exploits descending slopes. This balance between exploration of the solution space and exploitation of the slopes leading to local minima is the key to a successful local search algorithm.

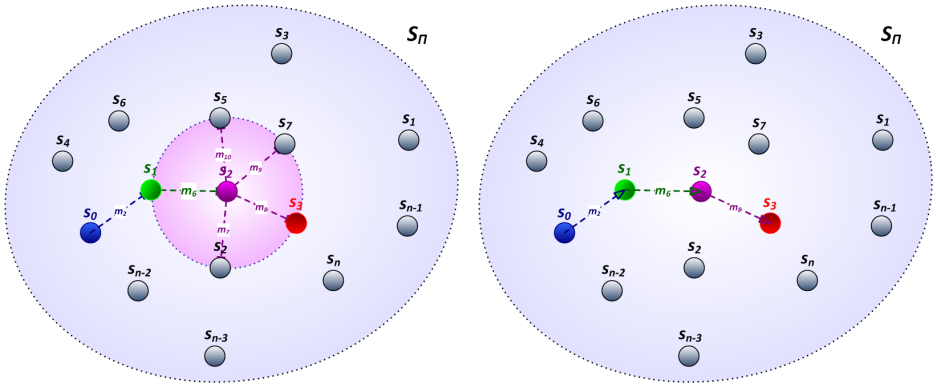
A potential solution comes in the form of *metaheuristics*, general problem-independent algorithmic frameworks that provide a guidance to build problem-specific heuristics. They incorporate general strategies to escape local minima and to navigate the search space and can be easily adapted to very different problems. There is a very rich literature on metaheuristics based on local search, and there is common consensus that



(a) Initial solution, move evaluation and selection



(b) Second move evaluation and selection



(c) Last move evaluation and selection

Figure 2.3: Local search path in the search space

using the local search paradigm inside a state-of-the-art metaheuristic constitutes a best practice. Widely used methods are Simulated Annealing (Kirkpatrick et al., 1983; Černý, 1985), Tabu Search (Glover, 1989, 1990), GRASP (Feo and Resende, 1995), Variable Neighborhood Search (Mladenović and Hansen, 1997), and Iterated Local Search (Lourenço et al., 2003). We refer to Glover and Kochenberger (2006) for a comprehensive review.

2.2 Multi-Neighborhood Search

We have understood that exploring the search space of a complex problem is not a trivial and straightforward task. Help comes in the form of a metaheuristic structure that provides with state-of-the-art methodologies to escape local minima and to navigate the search space. However, the metaheuristic alone doesn't guarantee the success of the search process and the quality of the final solution. Navigating the search space with a single local search neighborhood might not be sufficient to prevent being trapped in local minima, basins of attraction, or plateaus. No metaheuristics can enhance the capability of exploring of the search space if the neighborhoods are not adequate for the purpose.

Multi-Neighborhood Search is the local search paradigm based on the composition of multiple neighborhoods. It enhances the search capabilities and overcomes the aforementioned limitations. A *multi-neighborhood* $\mathcal{N}_\cup$ is defined as:

$$\mathcal{N}_\cup = \bigcup_{k=1}^{K} \mathcal{N}_k \tag{2.2}$$

where $\mathcal{N}_1, \ldots, \mathcal{N}_K$ are the individual neighborhoods, and K is the number of neighborhoods that compose $\mathcal{N}_\cup$. Given that this multi-neighborhood is obtained by the union of the individual neighborhoods, we also talk about *union neighborhood*.

The motivations to compose multiple neighborhoods together are many. First of all, a local minimum, a plateau or a basin of attraction of a neighborhood is not necessarily a local minimum, a plateau or a basin of attraction of another one. Consequently, the multi-neighborhood reduces the likelihood of the search getting trapped in unpromising regions of the search space. Additionally, the composition of multiple neighborhoods shapes a larger neighborhood, which enhances the connectivity in the search space. That is, the probability to find a path from a solution to another is higher. Finally, a portfolio of neighborhoods makes the search more robust against

variations in the features of the instances and in the characteristics of the search space.

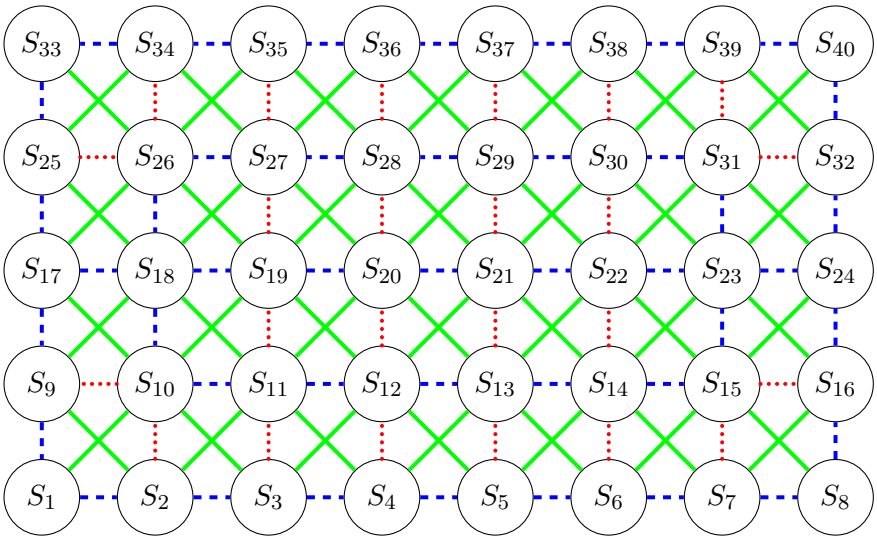


Figure 2.4: Connectivity of different neighborhoods in the search space.

Fig. 2.4 illustrates how Multi-Neighborhood Search enhances the connectivity in the search space. The small search space in the example has 40 solutions, labeled from S_1 to S_{40}. We have three neighborhoods, distinguished by three colors: *red* (dotted line), *blue* (dashed line) and *green*. If a solution S_j is a neighbor of solution S_i, an edge with the color of the corresponding neighborhood connects the two solutions. We notice immediately that only the blue neighborhood connects completely the search space. Despite what one might think at first sight, the green neighborhood is constrained by the choice of the initial solution. For example, if we start the search from solution S_{20}, solutions S_{12}, S_{19}, S_{21}, and S_{28} cannot be reached in any way using the green neighborhood alone. Therefore, one could decide to use the blue neighborhood. However, it is very inefficient. For example, to move from solution S_{13} to solution S_5, it requires at least nine moves, and this path may contains plateaus, local minima, and other traps. If, instead, we use the red neighborhood, one move is enough, and two moves suffice to a combination of the blue neighborhood with the green one. Therefore, the best choice in this scenario is to explore the search space with a combination of the *red*, the *blue* and the *green* neighborhoods in a multi-neighborhood, that provides all the possible paths to navigate the search space.

An ideal requisite is that the neighborhoods that compose the

multi-neighborhood are disjoint, that is, for any given solution S, $\mathcal{N}_i(S) \cap \mathcal{N}_j(S) = \varnothing$, $\forall \mathcal{N}_i, \mathcal{N}_j \in \mathcal{N}_\cup$. Even though it is not always possible to ensure that the intersection gives the empty set, an ideal multi-neighborhood contains as few repeated neighbors as possible. If multiple neighborhoods contain identical neighbors, the exploration is biased, and, by sampling the same neighbor more often than desired, leads to a waste of computational resources.

2.2.1 Exploration and move selection

Analogously to the single-neighborhood case, the multi-neighborhood exploration can be sequential or stochastic.

In *sequential exploration*, the neighborhoods are sorted and are explored sequentially. Inside each neighborhood, the neighbors are also explored sequentially.

In *stochastic exploration*, every neighborhood $\mathcal{N}_k \in \mathcal{N}_\cup$ is assigned with a probability $\sigma_k \in [0, 1]$ such that $\sum_{k=1}^{K} \sigma_k = 1$. The probabilities σ_k are referred to as *rates* or *weights*. The move choice is performed with the *two-step move selection*. First, we choose a random neighborhood, according to the probability vector, with a biased random selection. After a neighborhood is chosen, the move inside the neighborhood is selected stochastically, according to the uniform distribution. The two-step move selection is resumed as follows:

1. Neighborhood selection: $\mathcal{N}_k \leftarrow RandomNH(\mathcal{N}_\cup, \{\sigma_1 \dots \sigma_K\})$
2. Move selection: $m \leftarrow RandomMove(S, \mathcal{N}_k)$

In this thesis, we focus on stochastic exploration. A motivation is that a multi-neighborhood is larger than a single neighborhood and sequential and exhaustive exploration can be computational expensive. Moreover, the idea of having a portfolio of neighborhoods that are explored sequentially has been exploited by other metaheuristics, like Variable Neighborhood Search (Mladenović and Hansen, 1997), while the idea of selecting them stochastically received less attention. In principle, it is possible to mix the two approaches, by using a sequential exploration of the neighborhoods and a stochastic exploration of the moves inside the neighborhood, and vice versa. However, these possibilities are out of the scope of this thesis.

2.2.2 Internal biases

In addition to the probabilities σ_k, the multi-neighborhood can be equipped with internal biases. They are used, for example, to distinguish between a full and a restricted version of a neighborhood, where the restricted version is designed in order to contain only elite moves.

The *internal bias* of a neighborhood is guided by a parameter $b \in [0, 1]$. If the neighborhood is selected, a biased random selection is done before the choice of the move: with probability b, we employ the restricted version of the neighborhood, and, with probability $1-b$, we employ the general version. Thus, the two-step move selection becomes a *three-step move selection*, that involves, in order: neighborhood, bias, move.

Within this thesis, examples of use of internal biases are found in Chapters 4 and 7.

2.2.3 Differential cost evaluation

A local search algorithm may evaluate millions or even billions of moves during its execution. Normally, many more moves are evaluated than accepted, and therefore, it is crucial to compute the difference $\Delta F(S_j, S_i)$ efficiently. A naive way to do it consists in simulating the move by building the new solution S_j, computing $F(S_j)$, and obtaining the difference $F(S_j) - F(S_i)$. However, this is not efficient, given that a new solution must be built at every move evaluation. Even if we had a way to build the new solution from the current one efficiently, the evaluation of the cost function alone can be an expensive operation. Therefore, a key factor for the speed of Multi-Neighborhood Search is the implementation of efficient differential cost evaluations, that evaluate the cost of a local search move without the need to perform it.

Consider an objective function given by the linear combination of c cost components. If we also allow violations of hard constraints in the search space, we add other h penalties. The resulting cost function used by the local search is $F(S) = H(S) + f(S)$, where $f(S) = w_1 f_1(S) + \cdots + w_c f_c(S)$ is the objective function of the problem and $H(S) = w_{c+1} H_1(S) + \cdots + w_{c+h} H_h(S)$ is the combination of weighted penalties for the hard constraint violations. Constants $w_1, \ldots, w_c$ and $w_{c+1}, \ldots, w_{c+h}$ are, respectively, the weights of the objective function and of the hard components. The computation of the cost of a new solution implies the recalculation of the $c + h$ cost components in $f(S)$ and $H(S)$.

If we solve the problem using a multi-neighborhood of size K, we can

speed up the cost evaluation by breaking down the differential cost evaluation of $\Delta F(S_j, S_i)$ into $K(c+h)$ delta cost evaluation functions, one for every combination of neighborhood and cost component. Recalling that, with the two-step move selection, we first select a neighborhood $\mathcal{N}_k$ and then a move $m \in \mathcal{N}_k$, the delta cost for move m is written as follows:

$$\Delta F_k(S_j, S_i) = \sum_{l=1}^{c} w_l \Delta f_{l,k}(S_j, S_i) + \sum_{p=1}^{h} w_{c+p} \Delta H_{p,k}(S_j, S_i) \tag{2.3}$$

where $\Delta f_{l,k}(S_j, S_i)$ and $\Delta H_{p,k}(S_j, S_i)$ are, respectively, the delta costs of the l-th cost component of $f(S)$ and the p-th cost component of $H(S)$, for the k-th neighborhood.

This may seem to have the same complexity of computing the original cost difference, but it presents several advantages. First of all, given that a local search move modifies the solution locally, we can compute these delta costs very efficiently, if we consider only the specific parts of the solution that are perturbed by the move. This is much more efficient than recomputing the cost of the whole solution. Additionally, it is possible that not all cost components are affected by the move, so that we have to compute fewer than $c+h$ delta costs, with the others being zero.

Consider, for example, the 2-opt neighborhood shown in Fig. 2.1. In this case, only the pairs of edges that are removed and inserted are affected by the move. We can compute the delta cost efficiently, by subtracting the cost of the two removed edges and adding the cost of the two inserted edges. We don't need to recompute the cost of the whole route, which involves a much larger number of edges.

Even though the differential cost evaluation is not usually treated in the literature about local search, because it is considered more an implementation detail rather than a research aspect, it is a crucial component for the efficiency of any Multi-Neighborhood Search algorithm and it should always be incorporated into its design. In all the problems discussed in Part II, we have implemented a differential cost evaluation technique, even when not explicitly specified.

2.2.4 Redundant solution representations

The efficiency of the delta cost evaluation is based on the assumption that many more moves are evaluated than they are accepted. The possibility to implement efficient delta cost evaluations, however, does not come for free in terms of solution structure and usually relies on the definition of redundant representations of the solution.

These redundant data structures are used to speed up the computation of delta costs. In particular, they allow constant time look-ups and reduce as much as possible searches of components in vectors and nested cycles. Every time a move is executed, all the redundant data structures are updated, which implies inevitably a computational overhead. However, if the number of move executions is much smaller than the number of move evaluations, it is still largely advantageous to implement as many redundant solution representations as required by the delta costs.

Similarly to the delta cost evaluation, we include redundant solution representations in all problems presented in Part II.

2.2.5 Deterministic sequences

Due to the complexity of many combinatorial problems, we don't always want all solution components to be modified as a consequence of a neighborhood choice. A way to restrict the search space is to decide that only certain solution components are modified by the neighborhood relations, while others are deterministically repaired after the application of the move.

An example of deterministic sequence to compute part of the solution after neighborhood moves, is provided in Chapter 6.

2.2.6 Repair chains

In complex neighborhoods, even identifying the solution components involved in the move can be a challenging task. Consider a neighborhood that modifies a subset $\{s_1, \ldots, s_k\}$ of solution components, not necessarily of fixed cardinality. Consider that a move in the neighborhood leads to an invalid solution if the subset is not constructed properly. This neighborhood is potentially very large and we cannot just select random moves, if the probability to select a valid move is low.

A *repair chain* is a deterministic procedure to compute the components modified by the move. We define the move only on a limited number of components, for example a single component s_1 or a couple of components $\{s_1, s_2\}$. Then, if the solution is invalid, we *repair* it by iteratively adding the components $s_3, \ldots, s_k$ to the subset, ensuring that each new component is added consistently with those already in the partial subset. We follow the procedure until we bring the move in a state of validity, or until there are no more components that can be added and we declare the move invalid.

The use of repair chains makes the move computation more expensive, but guarantees that only valid moves are generated, consistently reducing

the overall computational cost of the neighborhood.

An example is provided in Chapter 5, where we solve the Sports Timetabling Problem using a multi-neighborhood composed of six neighborhoods, three of which are complex and require the use of repair chains.

2.3 Multi-Neighborhood Simulated Annealing

Like classic local search, also Multi-Neighborhood Search requires the guidance of an algorithm. Given that our approach is designed for stochastic exploration, a natural candidate is Simulated Annealing (SA).

SA relies on a parameter $T \in \mathbb{R}$ called *temperature*, that guides the acceptance rate of worsening moves. At each iteration, a move m is sampled at random and it is accepted according to the Metropolis criterion, which implies that improving and sideways moves are always accepted, while worsening moves are accepted with probability $e^{-\Delta F/T}$. The temperature T starts from an initial value T_0, and decreases following a *geometric cooling* scheme, until a final value T_f $(0 < T_f < T_0)$. Therefore, the cooling scheme reduces the likelihood of acceptance of worsening moves during the search.

Many variants of the SA procedure have been proposed in the literature, but along this thesis we use its basic version, as originally proposed by Kirkpatrick et al. (1983), with some minor arrangements described in this section. We forward the reader to Franzin and Stützle (2019) for a comprehensive and in-depth review of the SA variants.

2.3.1 Initial solution

Local search always starts from an initial valid solution $S \in \Sigma$, and SA is no exception. The initial solution can be generated using a random procedure that assigns random values to the solution components in a controlled manner, or by a probabilistic or deterministic greedy procedure. Alternatively, one can initialize the SA procedure with the best solution found by an external solver or from previous runs of the same SA procedure. This is known as *warm start*.

We will not delve into the details of the initial solution generation since it is not among the critical components in Simulated Annealing (see Franzin and Stützle, 2019). Moreover, in Chapter 5, we show that a comparison of greedy and random initial solution strategy for the Sports Timetabling Problem does not significantly influence the performance of the algorithm.

2.3.2 Acceptance criterion

A key component of SA is the Metropolis acceptance criterion (Metropolis et al., 1953). After a move m with cost cost $\Delta F > 0$ is sampled, a random real number is drawn in the interval $(0, 1)$. If its value is below $e^{-\Delta F/T}$, the move m is accepted, otherwise it is rejected. If $\Delta F \leq 0$, the move is always accepted.

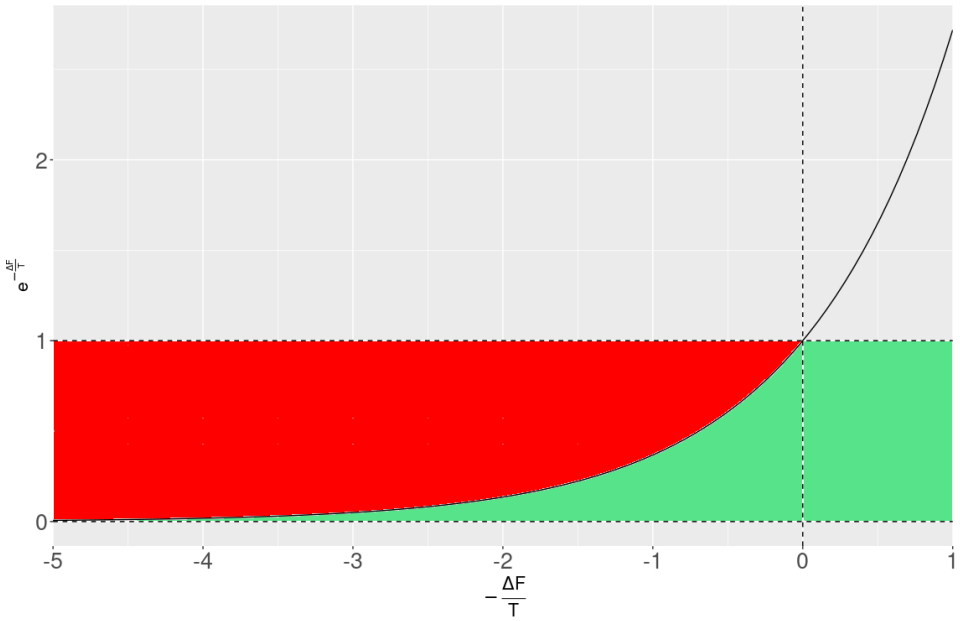


Figure 2.5: Metropolis acceptance criterion in Simulated Annealing.

Fig. 2.5 shows the value $-\Delta F/T$ on the x-axis and, on the y-axis, the acceptance probability $e^{-\Delta F/T}$. For every value of $-\Delta F/T$, the acceptance probability is the height of the curve in that point. The area painted in green represents the area where moves are accepted, while the area painted in red corresponds to rejected moves. When $-\Delta F/T < 0$, the value $e^{-\Delta F/T}$ is comprised between 0 and 1 and defines the acceptance probability. It decreases as we go towards greater negative values, which are given by either lower temperature T or higher positive ΔF.

2.3.3 Cooling scheme and cut-off

The temperature starts from value T_0, and reaches the final value T_f, evolving through intermediate temperatures $T_1, T_2, \ldots, T_{f-1}$. A parameter

N_s defines the number of sampled moves at each temperature level (s stands for sampled). A variable n_s counts the number of moves sampled at the current temperature. When counter n_s hits the value N_s, the temperature cools down according to the geometric cooling scheme $T_{k+1} = \alpha T_k$, where $\alpha \in (0, 1]$ is the cooling rate. Afterwards, n_s is reset to zero.

Knowing T_0, T_f, and α, we can determine the number of temperatures f in a SA run. Given that $T_f = T_0 \cdot \alpha^f$, we obtain that

$$f = log_\alpha \left(\frac{T_f}{T_0} \right)$$

is the number of temperatures in a Simulated Annealing run.

In order to speed up the early stages of the search, we adopt the so-called *cut-off* mechanism (see Johnson et al., 1989), which decreases the temperature when a fixed number of moves N_a has been accepted (a stands for accepted). The intuition behind this is that there's no need to remain at a given temperature level if most of the worsening moves are accepted, as it is a symptom of random roaming in the search space. We use a counter n_a to record the number of accepted moves in the temperature. In case n_a reaches the value N_a while $n_s < N_s$, the cut-off is applied and we decrease the temperature. We redistribute the $N_s - n_s$ iterations what would be lost among the remaining temperatures, adapting N_s after the cooling step, as follows:

$$N_s \leftarrow N_s + \frac{N_s - n_s}{log_\alpha \left(\frac{T_f}{T} \right)}$$

where the quantity $log_\alpha(T_f/T)$ is the number of remaining temperatures, computed at the current temperature T.

In order to ease the determination of N_a, we define the parameter $\rho = N_a/N_s$ (with $0 < \rho \leq 1$) which represents the fraction of the initial number of iterations per temperature N_s that must be accepted to apply the cut-off. Therefore, $N_a = \rho \cdot N_s$. Nevertheless, as N_s is adapted during the search to save iterations, the value of N_a is still considered as an independent parameter and it is not updated.

2.3.4 Stopping criterion

We have previously seen how to compute the number of temperatures f in a Simulated Annealing run. Given N_s, we obtain that the total number of local search moves evaluated $\mathcal{I}$ is:

$$\mathcal{I} = f \cdot N_s$$

In general, however, we want to decide the length of the run beforehand, and therefore we set the value $\mathcal{I}$. Given that also T_0, T_f and α are given as input parameters, N_s is computed as a dependent parameter, as follows:

$$N_s = \frac{\mathcal{I}}{f} = \frac{\mathcal{I}}{log_\alpha\left(\frac{T_f}{T_0}\right)}$$

We point out that the use of $\mathcal{I}$ as length of the algorithm run guarantees the reproducibility of the experiments, and all the works presented in Part II follow this approach. Conversely, the works presented in Part IV use a fixed runtime. The motivations for this choice and its implications on reproducibility are discussed in the respective chapter.

2.3.5 Algorithm description

The Multi-Neighborhood Simulated Annealing (MNSA) that uses the algorithm components described in this chapter is presented in Fig. 2.6. We provide hereafter its detailed description, given that it is the baseline metaheuristic employed in the work presented in Part II.

It takes as input the search space Σ, a multi-neighborhood $\mathcal{N}_\cup = \bigcup_{k=1}^{K} \mathcal{N}_k$, the total iterations $\mathcal{I}$, a cost function F, values for the parameters T_0, T_f, α, ρ, and the neighborhood rates $\sigma_1 \dots \sigma_K$.

The algorithm starts at line 1 with the initialization of the temperature T to T_0 and the computation of the values N_s and N_a. At line 2 the initial solution is generated, which is also assigned as the best solution found so far.

The main loop from lines 3 to 20 is executed until the temperature doesn't go below the final value T_f. Line 4 initializes the counters n_s and n_a of sampled and accepted neighbors in the current temperature. Lines from 5 to 18 contain the inner loop that is repeated to sample up to N_s neighbors, or up to N_a accepted moves.

The multi-neighborhood exploration takes place at lines 6–7, according to the two-step move selection approach.

The move delta cost ΔF is computed at line 8, using the efficient delta cost computation. In case $\Delta F \leq 0$ (line 9) the move is always accepted (line 10) and the counter n_a is incremented (line 11). Furthermore, at lines 12–13 a potential new best solution is detected and the best-so-far is updated. A worsening move, instead, is accepted only if it respects the Metropolis criterion (lines 15–17). Finally, the counter of the number of sampled neighbors is updated at line 18.

```
procedure MULTI-NEIGHBORHOOD SIMULATED ANNEALING
          (SearchSpace Σ, Multi-Neighborhood 𝒩∪, Iterations ℐ,
          CostFunction F, Parameters T0, Tf, α, ρ, {σ1 ... σK})
1:    T ← T0, Ns = ℐ/log_α (Tf/T0), Na = ρNs
2:    S ← RandomState(), Sbest ← S
3:    while T ≥ Tf
4:        ns ← 0, na ← 0
5:        while ns < Ns ∧ na < Na
6:            𝒩k ← RandomNH(𝒩∪, {σ1 ... σK})
7:            m ← RandomMove(S, 𝒩k)
8:            ΔF ← F(S ⊕ m) − F(S)
9:            if (ΔF ≤ 0)
10:               S ← S ⊕ m
11:               na ← na + 1
12:               if (F(S) < F(Sbest))
13:                   Sbest ← S
14:           else
15:               if (RandomReal(0, 1) < e^(−ΔF/T))
16:                   S ← S ⊕ m
17:                   na ← na + 1
18:           ns ← ns + 1
19:       T ← T · α
20:       Ns → Ns + (Ns − ns) /log_α (Tf/T)
21:   return Sbest
```

Figure 2.6: Multi-Neighborhood Simulated Annealing procedure

After exiting the inner loop, the temperature is updated at line 19 according to the geometric cooling scheme, that depends on the parameter α. If the inner loop was completed because the number of accepted neighbors reached the threshold N_a, the unused $N_s - n_s$ move evaluations are redistributed equally among the remaining temperatures (line 20). The outcome of the algorithm is the best solution found S_{best} (line 21).

2.4 Parameter tuning

The multi-neighborhood needs input values for the neighborhood rates $\sigma_1, \ldots, \sigma_K$, and the possible internal biases b_k. Moreover, the SA has its own parameters: the start temperature T_0, the final temperature T_f, the cooling rate α, and the cut-off ratio ρ.

In general, we can safely assume that the values of the

multi-neighborhood parameters and those of SA are independent between each other. Even though this is not true in a strict sense, it allows to tune them separately, simplifying the parameter space. We employ this approach in all the works presented in Part II.

Regarding SA parameters, the values of T_0 and T_f are related with the average delta costs encountered during the search, as both the temperature and the delta cost appear in the Metropolis acceptance criterion. Conversely, α and ρ are less affected by the features of the instance.

An appealing characteristic of the multi-neighborhood approach would be its ability to self-adjust, meaning that it can autonomously adapt the neighborhood rates during the search. We explore this in Part IV, where we propose an online tuning mechanism for $\sigma_1, \ldots, \sigma_K$ through a reinforcement learning approach. In this way, we reduce the computational cost of the tuning procedure and we do not have to assume independence with the SA parameters. At the same time, this makes the multi-neighborhood robust against changes in the instance landscape.

Finally, we underline the importance of always conducting a statistically principled tuning procedure, minimizing premature commitment in choosing parameter values. To do so, we employ automated algorithm configuration tools that incorporate state-of-the-art techniques to sample points in the parameter space and to evaluate configuration on the basis of statistical tests, conducted on the experimental results.

2.5 Further combinations of neighborhoods

This thesis focuses on the *union neighborhood*, which is the multi-neighborhood obtained by the union of individual neighborhoods. It is not the only possible way to combine neighborhoods together, and, even if it is not in the scope of this thesis, we briefly discuss here some alternatives.

The *product neighborhood* is obtained by the sequential application of two neighborhoods and it is defined as $\mathcal{N}_{1\times 2} = \mathcal{N}_1 \times \mathcal{N}_2$. The advantage of applying two neighborhood moves at once resides mostly in its possibility to perform longer jumps. Imagine that we have two neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ and that exists a path in the search space $S_i \to S_j \to S_k$, with $S_j \in \mathcal{N}_1(S_i)$, $S_k \in \mathcal{N}_2(S_j)$. Consider also that $\Delta f(S_k, S_i) < 0$, and that S_k is not reachable from S_i with any of the neighborhood relations and that $\Delta f(S_j, S_i) > 0$. In this case, it might be that the move that brings from S_i to S_j is never performed because it is worsening, and the promising path from S_i to S_k is never walked. The neighborhood $\mathcal{N}_{1\times 2}$ contains the improving move that

brings from S_i to S_k, which has a delta cost $\Delta f(S_k, S_i) < 0$.

The *Cartesian neighborhood* is the union of all the product neighborhoods. It is defined as follows:

$$\mathcal{N}_{\prod} = \bigcup_{i,j=1,i\neq j}^{K} \mathcal{N}_i \times \mathcal{N}_j \tag{2.4}$$

We remark that a Cartesian neighborhood is much larger than a union neighborhood. A Cartesian neighborhood made up by K neighborhoods contains $K(K-1)$ product neighborhoods, that becomes K^2 if we allow $i = j$ in Eq. (2.4). Moreover, we don't want to lose the possibility to employ individual neighborhoods, which adds other K neighborhoods.

There are other ways of using multiple neighborhoods that are not strictly multi-neighborhoods. One option is the *token ring*, which involves alternating between neighborhoods each time a local minimum is encountered. The token ring at the basis of Variable Neighborhood Search (Hansen et al., 2017). However, we do not classify it as a multi-neighborhood according to our definition, because it only connects the neighbors associated with the currently active neighborhood. This is in contrast to the union and Cartesian neighborhoods, which allow to reach all neighbors in all available neighborhoods at every step of the search. Despite this distinction, the token ring is a highly effective approach, and in particular its use within the VNS metaheuristic has yielded remarkable results on a large class of combinatorial problems.

Chapter 3

Construct, Merge, Solve, and Adapt

Construct, Merge, Solve & Adapt (CMSA) is a general matheuristic for combinatorial optimization originally proposed by Blum et al. (2016), based on the idea of constructing a reduced instance of the full problem by means of merging solution components obtained, for example, by the repeated execution of a randomized construction heuristic, and the solution of the reduced instance by means of an exact solver. The algorithm is also equipped with an aging mechanism to ensure that unpromising solution components are discarded after a certain number of iterations. Existing applications of CMSA encompass various hard combinatorial optimization problems. We mention, among many, the multi-dimensional knapsack problem (Blum and Ochoa, 2021), prioritized test data generation (Ferrer et al., 2021), and refueling and maintenance planning of nuclear power plants (Dupin and Talbi, 2021). Extensions include the Self-Adaptive CMSA (Akbay et al., 2022) and the Multi-Constructor CMSA, that we introduce in Chapter 9 of this thesis. In the following sections, we provide an overview of the general method, adopting the terminology from Blum et al.. Moreover we give a brief introductory discussion of the novel Multi-Constructor CMSA and we discuss the parameter tuning for CMSA.

3.1 The CMSA Algorithm

We define a set $\mathcal{C}$ to encompass all conceivable components that could make up solutions in a combinatorial problem. In this context, a valid solution S is represented by a subset of the solution components in $\mathcal{C}$, denoted as $S \subseteq \mathcal{C}$.

On the majority of combinatorial problems, $\mathcal{C}$ is large, and finding good or optimal solutions might be challenging even with modern MIP solvers. Nevertheless, if we had a way to build a restricted set $\mathcal{C}' \subseteq \mathcal{C}$, which only contains promising solution components, we could affordably find a proven optimal solution on $\mathcal{C}'$. In this context, we say that $\mathcal{C}'$ is a *sub-instance* of $\mathcal{C}$.

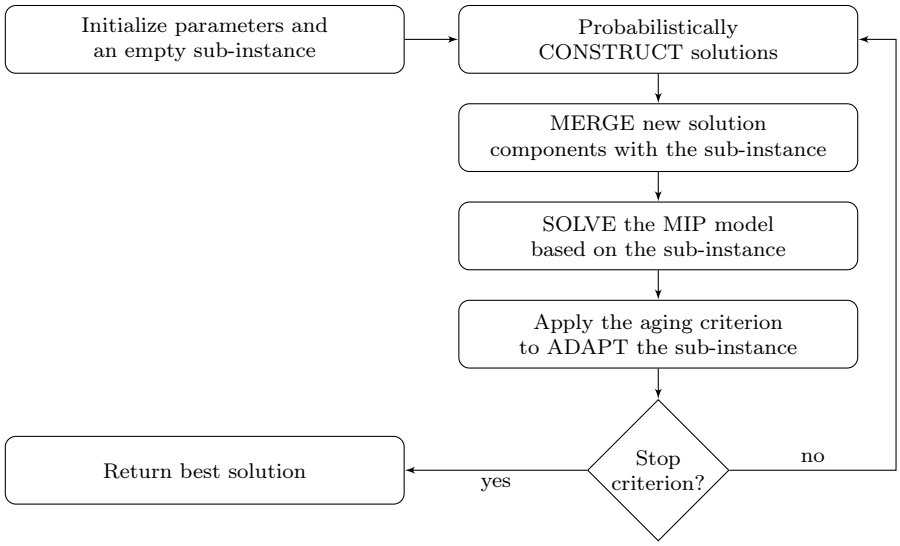


Figure 3.1: General CMSA flow

The CMSA algorithm, represented in Fig. 3.1, is composed by four iterated phases:

Construct: A number of feasible solutions are constructed.

Merge: The solution components from these solutions are merged into $\mathcal{C}'$.

Solve: A (possibly) exact solver is applied to the sub-instance $\mathcal{C}'$.

Adapt: $\mathcal{C}'$ is adapted by removing unpromising solution components.

A key factor for a successful application of CMSA is the ability to keep the sub-instance into reasonable sizes: it shouldn't be too small, because it would not contain enough diverse components to find good solutions, but it shouldn't be too large either, because it would be too hard to solve and the exact solver might not find any good solution. The size of the sub-instance in relation to the size of the problem instance depends on several factors,

including the nature of the constructive heuristic, the number of solutions generated at each iteration, and the aging mechanism.

When we have an integer linear or non-linear programming model for the problem, one way to define the set of solution components is by considering each combination of a variable with one of its values as a solution component.

In the following, we discuss the four phases of the CMSA algorithm on the assumption that we have an integer linear model of the problem in the form:

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} && \text{(3.1a)} \\ \text{subject to:} \quad & A\mathbf{x} \geq \mathbf{b} && \text{(3.1b)} \\ & \mathbf{x} \in \{0,1\}^n && \text{(3.1c)} \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of binary variables, $\mathbf{c} \in \mathbb{R}^n$ is the vector of coefficients of the objective function and $\mathbf{b} \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$ are a vector and a matrix that define the constraints. For a given $x \in \mathbf{x}$ we define two solution components: $x = 0$ and $x = 1$.

3.1.1 CONSTRUCT: construction procedure

In the CONSTRUCT phase, we call *constructor* the algorithm responsible for generating n_{sols} feasible solutions. A reasonable strategy is to employ, in the role of constructor, a probabilistic constructive heuristic, such as a randomized greedy algorithm. Greedy algorithms start with an empty solution and iteratively add components to the partial solution based on a problem-specific greedy function, until a complete solution is built.

A deterministic greedy can be suitably randomized using a *deterministic rate* parameter $d_{rate} \in [0, 1]$ and a *candidate list*, with size $c_{list} \in \mathbb{N}$. Before every insertion of a new component in the partial solution, a random number is drawn between 0 and 1. If the number falls below d_{rate}, the step is taken deterministically, and the best candidate component according to the greedy function is added to the solution. Otherwise, a list containing the c_{list} candidate components that score best in the greedy function is built, and a random component is chosen uniformly from the list.

It is worth noting that introducing randomness, especially early in the construction process, can lead to significant variation in the resulting solutions, even when using high deterministic rates.

3.1.2 MERGE: sub-instance

A sub-instance $\mathcal{C}'$ is composed by all the solution components that have been generated at least once by the construction procedure, and haven't been eliminated yet because of aging.

In the n_{sols} solutions coming from the CONSTRUCT phase, each variable $x \in \mathbf{x}$ has taken either always the value 1 in all solutions, or always the value 0, or 0 in certain solutions and 1 in others. If a variable has taken both values 0 and 1, than both the components $x = 0$ and $x = 1$ are added to the sub-instance. If the variable has taken only value 1 or only value 0, then only the corresponding component is added to the sub-instance. Except from the first iteration, the sub-instance is not empty and therefore the components are merged with the existing ones.

A possible variation is to include always the component $x = 0$, even if a given variable x is consistently found with value 1 by the constructor.

3.1.3 SOLVE: exact solver

A solver is called to find a solution using solely the components in the sub-instance. The intuition is that we can discover solutions that are better than the best solution generated by the repeated execution of the constructor.

First, we fix in the model the variables $x \in \mathbf{x}$ for which only the component $x = 0$ or only the component $x = 1$ is present in the sub-instance. This is done, for example, by imposing the constraints $x = 0$ or $x = 1$. We say that these are the *fixed variables*. The variables that present both components $x = 0$ and $x = 1$ are not constrained and are free to take any value in the domain $\{0, 1\}$. We call them *released variables*.

The reduced model that we obtain is smaller than the original model. A MIP solver that was unable to solve the original model, might be able to solve the reduced one in a reasonable running time. It might still be convenient, however, to impose a time limit t_{exc} on the solver, for example to prevent long run times required just to prove the optimality of a solution, at the expenses of experiment reproducibility. Therefore, the outcome of the SOLVE phase can be either a proven optimal solution on the sub-instance, or the best solution found within the time limit t_{exc}.

A best practice is to perform a *warm start*, which provides the solver with the best-so-far solution.

3.1.4 ADAPT: aging

While CMSA shares some features in common with other matheuristics like LNS (Shaw, 1998) and Kernel Search (Angelelli et al., 2010), a key difference from other existing methods is its aging mechanism. The intuition behind the aging mechanism of CMSA is that only profitable solution components should be kept into the sub-instance in the long term, where profitable means that they are chosen by the solver in the SOLVE phase. Conversely, the presence in the sub-instance of components that are consistently discarded by the exact solver do not provide any advantage, but rather constitutes a disturbing computational load.

The aging mechanism is based on the concept of *age* of a solution component, which is a non-negative integer value associated to all solution components in the sub-instance. For every new component $x = 0$ or $x = 1$ that we merge in the sub-instance, we set the counter of its age to zero. If the MIP solver finds a solution with $x = 0$ or $x = 1$, we reset the age of the corresponding component to 0. Otherwise, we increase the age of the component by one. When the age of a component reaches the threshold age_{limit}, the component is removed from the sub-instance.

The aging mechanism plays a crucial role in maintaining the size of the sub-instance reasonable, which is vital because MIP solvers are highly sensitive to the size of the input. The sub-instance size is guided by the parameter age_{limit}, in relation also with n_{sols}, d_{rate}, and c_{list}. Naturally, the sub-instance is considered "small" or "large" in relation to the capabilities of the exact solver to solve it within the granted computational time t_{exc}. In general, increasing n_{sols} and age_{limit} leads to a larger sub-instance size, while a higher d_{rate} reduces it. Finding a good balance among these values is important to ensure the performance of CMSA.

3.2 Multi-Constructor CMSA

The novelty that we introduce in this thesis is the Multi-Constructor CMSA, which extends the original CMSA by allowing the use of multiple independent *constructors*, into the CONSTRUCT phase. Differently from the original CMSA, that uses a single construction procedure, every solution is generated by a constructor chosen from the pre-defined portfolio of constructors. The intuition is that a diversity in the constructors leads to a higher diversity in the solution components in the sub-instance $\mathcal{C}'$. More importantly, it might be that certain solution components are not accessible to a given constructor because of the nature of its constructive procedure, and if a component is

never generated it cannot be found in the sub-instance. Therefore, using multiple constructors, we can access areas of the search space that are not reachable by means of a single constructor.

Leveraging the principles of Multi-Neighborhood Search (Chapter 2), we propose a stochastic selection criterion for the constructors in Multi-Constructor CMSA. In order to bias the choice towards the most promising constructors, *probabilities*, also named *weights* or *rates*, are associated with them. Given a Multi-Constructor composed by K constructors $\mathcal{H}_1, \ldots, \mathcal{H}_K$, the probabilities are parameters $\sigma_1, \ldots, \sigma_K$ such that $\sigma_i \in [0, 1]$ and $\sum_{i=1}^{K} \sigma_i = 1$.

procedure MULTI-CONSTRUCTOR CMSA
(Problem instance, values for parameters
n_{sols}, age_{limit}, t_{exc}, probabilities $\{\sigma_1, \ldots, \sigma_K\}$)
$S_{bsf} \leftarrow$ NULL
$\mathcal{C}' \leftarrow \emptyset$
while stop criterion not met
for $i = 1, ..., n_{sols}$
$\mathcal{H}_k \leftarrow$ ChooseConstructor$(\{\mathcal{H}_1, \ldots, \mathcal{H}_K\}, \{\sigma_1, \ldots, \sigma_K\})$
$S \leftarrow$ ProbabilisticSolutionGenerator$(\mathcal{H}_k, \mathcal{C})$
for all $c \in S$ and $c \notin \mathcal{C}'$
$age[c] \leftarrow 0$
$\mathcal{C}' \leftarrow \mathcal{C}' \cup \{c\}$
$S_{opt} \leftarrow$ ApplyExactSolver$(\mathcal{C}', t_{\text{exc}})$
if S_{opt} is better than S_{bsf}
$S_{bsf} \leftarrow S_{opt}$
for all $c \in \mathcal{C}'$
if $c \in S'_{opt}$
$age[c] \leftarrow 0$
else
$age[c] \leftarrow age[c] + 1$
if $age[c] \geq age_{limit}$
$\mathcal{C}' \leftarrow \mathcal{C}' \setminus c$
return S_{bsf}

Figure 3.2: Multi-Constructor CMSA

The pseudo-code of the Multi-Constructor CMSA is presented in Fig. 3.2. The algorithm takes as input a problem instance, the number of solution constructions per iteration n_{sols}, and a maximum allowed age for the solution components age_{limit}. It returns a solution S_{bsf}, which is the best solution found during its execution.

The algorithm begins with the initialization of the best-so-far solution S_{bsf} to NULL and the sub-instance $\mathcal{C}'$ to an empty set, at lines 1 and 2. The main loop is comprised in the lines from 3 to 19, and continues as long as the stop criterion is not met. A suitable stop criterion is the total number of CMSA iterations.

At each iteration, n_{sols} feasible solutions are probabilistically generated in the CONSTRUCT phase. Here is the main difference with respect to the original CMSA. At every solution generation, a constructor $\mathcal{H}_k$ is chosen at random at line 5, according to the probabilities $\sigma_1, \ldots, \sigma_K$, from the pool of K constructors $\mathcal{H}_1, \ldots, \mathcal{H}_K$. Then, a solution is generated with the chosen constructor (line 6).

The solution components associated with these solutions are subsequently merged into $\mathcal{C}'$ (MERGE phase, lines 7–9). The age of the new components added to the sub-instances is set to zero (line 8).

An exact solver is then applied to optimize $\mathcal{C}'$ (SOLVE phase, line 10), possibly within a given time limit t_{exc}. The best-so-far solution is updated if the solution found is better than the current best-so-far solution (lines 11–12).

Finally, in lines 13–19, the so called ADAPT phase takes place. First of all, the age of all solution components found in S_{opt} is set to zero (lines 14–15), while, the age of solution components in $\mathcal{C}' \setminus S_{opt}$ is increased by one (line 16–17). Moreover, the solution components in $\mathcal{C}'$ that have reached the maximum age (age_{limit}) are removed from the sub-instance (lines 18–19).

There are many advantages that come from using multiple constructors in CMSA. First of all, constructors can be of different nature. For example, a greedy constructor that builds complete solutions and, therefore, guarantees that at least a valid solution is present in the sub-instance, can be used together with constructors that only build partial solutions. Additionally, while it is not trivial to come up with different constructors for the same problem, that are together high-quality and diverse, in many cases there exist already in the literature efficient greedy algorithms that can be easily randomized and used as constructors.

In conclusion, the Multi-Constructor CMSA is an extension of CMSA that provides a general strategy for using different constructors within the CMSA procedure. The introduction of the Multi-Constructor CMSA, along with its initial application, can be found in Chapter 9, where we adapt six greedy algorithms from the literature as constructors to solve the Maximum Disjoint Dominating Sets Problem.

3.3 Parameter tuning in CMSA

Several parameters have been introduced in this chapter for CMSA. We have the number of solutions generated at each iteration n_{sols}, the maximum age of solution components age_{limit}, which are strictly CMSA parameters. Additionally, we set a time limit for the exact solver t_{exc}. Furthermore, we consider the parameters of the construction procedure that are the deterministic rate d_{rate} and the candidate list size c_{list}. Finally, the probabilities of the constructors $\sigma_1, \ldots, \sigma_K$ are specific parameters of the Multi-Constructor CMSA. As mentioned before, it is important that the size of the sub-instance is kept reasonable, in order to allow the exact solver to find a solution in the given time limit, and this makes parameter tuning a critical task. Moreover, if for a certain problem the instances are very different in size, we have to keep into account that a single set of parameter values might be ineffective and we might need to tune the parameters separately for size ranges, or to include a feature-based relationship that links the parameters to the size of the instance.

Part II

Applications of Multi-Neighborhood Search

Chapter 4

Minimum Interference Frequency Assignment

Allocating radio spectrum resources is a crucial problem in the design and operation of mobile communication networks. In particular, the Frequency Assignment Problem (FAP) consists in assigning, in an efficient way, a limited number of frequencies to communication links (Murphey et al., 1999). In this chapter, among all the possible frequency assignment models, we consider the Minimum Interference Frequency Assignment Problem (MI-FAP), which is the most studied model, mostly due to its practical importance. The MI-FAP aims at allocating frequencies so that interferences between adjacent frequencies in geographically close links are minimized.

We undertake the two versions of the problem proposed in the COST 259 project (Correia, 2001) and by Montemanni et al. (2001), respectively, as they both come along with a very challenging dataset, that have been used as benchmark for many studies.

Even though the technology has evolved substantially since these formulations have been proposed, the frequency assignment problem remains relevant on more recent environments, such as 5G (Lin et al., 2015), edge computing (Zhang et al., 2020), D2D networks (Zhao et al., 2018), and military applications (Wang and Henz, 2017; Lal et al., 2018). However, no specific new formulation and benchmarks have emerged so far, therefore these datasets remain the most popular and challenging benchmarks.

For the solution of this problem we have designed and implemented a Multi-Neighborhood Search approach based on a suitable combination of different neighborhoods, driven by a Simulated Annealing (SA) procedure. SA has been used also by many other authors for FAP with good results,

suggesting that it is particularly suitable for this type of problems (see Section 4.2 on Related work).

In order to obtain the best configuration of the algorithm for the specific problem versions and datasets, we have tuned the algorithm using a comprehensive and statistically-principled tuning procedure.

We also performed an extensive experimentation with the aim of comparing our results with previous literature. Although a totally precise comparison is not possible, the outcomes of the experiments show that our solution method is able to outperform all those reported for the version of Correia (2001) for most of the instances and to reach the same level of the best results for the version of Montemanni et al. (2001), on the respective datasets. In addition, we have reached many new best-known solutions for the COST 259 dataset.

Finally, in order to foster future comparisons, we published our solutions on the web. To this aim, we created a novel data format for both input and output files based on JSON. All data is available for inspection and comparison at `https://opthub.uniud.it`. Our repository is conceived in such a way that also other researchers can validate and upload their solutions, so that they become immediately available for everybody along with the timestamp of the upload.

4.1 Problem definition

For the sake of completeness, we describe here informally the two versions of the MI-FAP problem, and we forward the reader to Correia (2001) and Montemanni et al. (2001) for more details on the specific formulations and for the precise mathematical models.

We proceed by firstly describing the problem version proposed within the COST 259 project, and then we move to the simpler variant proposed by Montemanni et al. (2001). In the following, for conciseness, we refer to the two versions as MI-FAP-I and MI-FAP-II, respectively.

The key elements of MI-FAP-I are the following ones:

Cells: a cell is an equipment that provides communication service to a given geographical area and has a fixed number of transmitters.

Transmitters: a transmitter is a single device that transmits the signal, and requires the assignment of a frequency. For each cell, one of the transmitters is designated to carry the control signal whereas the others carry traffic signals. The control signal needs special treatment

in terms of *separation constraints*, as explained below (see *handover* separation rule).

Frequencies: a fixed number of frequencies (or channels) are available, each one identified by an integer value. Frequencies that are adjacent in terms of actual transmission bandwidth are assigned to consecutive values. Some frequencies can be forbidden either for a given cell or globally for the entire area.

Sites: a site is a physical installation where several cells (typically three) are located.

In order to avoid disturbance in the communication, the assignment of frequencies to transmitters has to satisfy a set of separation rules:

Co-cell: transmitters belonging to the same cell must have a given frequency separation. The typical separation value is 3 and it is defined at global level, but some cells can have specific (and different) separation requirements.

Co-site: transmitters belonging to cells at the same site must have a given frequency separation, typically 2.

Handover: cells that are geographically adjacent might suffer from the so-called handover effect, therefore requiring a specific separation among the transmitters of those cells on the basis of their specific type (i.e., control / traffic). Typically, there must be a separation of 2 between the control channels, a separation of 1 or 2 between a control channel and a traffic channel, and a separation of 1 between traffic channels.

Additional separations: ad hoc separations between pairs of cells due to specific conditions might be required, and they have to be enforced on all pairwise combinations of transmitters of the two cells involved.

In addition to these mandatory channel separations, which must be always fulfilled (i.e., they are so-called *hard constraints*), also a milder interference of frequencies between cells has to be taken into account in terms of an objective function to be minimized. In detail, for each pair of cells the *interference cost* that occurs in case of assignment to the same frequency (*same-channel* interference) or to adjacent (i.e., whose distance

is 1) frequencies (*adjacent-channel* interference) is specified as a pair of real-valued numbers.

When the cells are geographically apart, there is no interference, so there will be no penalty for assigning the same frequency or an adjacent one. It might also be possible that only the same-channel interference is relevant and the adjacent-channel interference is zero (the opposite is obviously not possible). In this version of the problem, no interference penalty is ever assigned in case of frequencies at distance 2 or more.

The specification for MI-FAP-II is much simpler, as there are no explicit notions of cell, site, and handover. Constraints are expressed directly at the transmitter level, by specifying the required separation between pairs of transmitters and the cost of its violation. The penalty for violating the separation is fixed, independently of the degree of violation. This is different from MI-FAP-I in which same- and adjacent-channel interferences are weighted differently. Finally, there is no explicit distinction between hard and soft constraints, though, in some instances, there are separations with an extremely high cost (10^8), that we interpreted as a hard one.

4.2 Related work

The literature on Frequency Assignment is very vast. For this reason, we focus our overview specifically on the Minimum Interference formulation. We refer to the comprehensive and enlightening survey by Aardal et al. (2007) for the general problem and to the FAP website (Eisenblätter and Koster, 2000) for other publications, benchmark instances, and results (unfortunately not very up-to-date).

The MI-FAP originates from the study by Allen et al. (1987) and was thereafter investigated in many other works (Duque-Antón et al., 1993; Kapsalis et al., 1995; Aardal et al., 1996; Crisan and Mühlenbein, 1998; Borndörfer et al., 1998; Koster et al., 1999; Tiourine et al., 2000; Koster et al., 2002; Björklund et al., 2005; Kolen, 2007), mainly in connection to the CALMA project (Aardal et al., 2002).

The MI-FAP-I formulation emerged within the COST 259 project "Wireless Flexible Personalised Communications" (Correia, 2001), which involved more than 200 European research institutions and companies in the area of mobile radio during the years from 1996 to 2000. One of the contributions of the project was a dataset of 32 realistic instances, which has become well-known and a very challenging benchmark for GSM network planning (see Section 4.4.1 for details about this dataset).

Over the years, MI-FAP-I has been mainly tackled by metaheuristic methods, because large-size scenarios of the COST 259 dataset are still beyond the reach of exact approaches. Among the metaheuristics methods, a Simulated Annealing (SA) approach for MI-FAP-I was firstly proposed by Beckmann and Killat (1999). This approach relies on a cell-based local search neighborhood with some restrictions: a cell is randomly selected, then the frequency of the transmitter (within the cell) with the largest interference cost is substituted with a new permitted frequency causing the smallest interference cost.

Hellebrandt and Heller (2000) apply a variant of SA, the *threshold accepting* algorithm, where a new solution is accepted if the deterioration of the value of the objective function is less than a given threshold, which is reduced during the search process. They implement a basic neighborhood (i.e., the one that changes the frequency to a single transmitter) but forbidding those moves that produce violations of the hard constraints. In addition, they also employ at each iteration a *one-cell reoptimization* process, by means of a dynamic program that performs a simultaneous exchange of all the frequencies assigned to a cell when this improves the current solution.

The dynamic programming method has been generalized to cliques of vertices by Mannino et al. (2007), who also employ a Simulated Annealing algorithm as their main search procedure. They also show that, for some restricted cases under some specific hypothesis on the subsets of transmitters, the MI-FAP can be reduced to a maximum weighted stable set problem, which is solvable in polynomial time. This theoretical result has been exploited to search effectively in a large-scale neighborhood, defined as the set of all transmitters whose frequency can be simultaneously replaced without incurring in any violation.

Montemanni et al. (2003) propose a Tabu Search procedure with a dynamic length tabu list in which the neighborhood relation changes the frequency of a single transmitter involved in at least one constraint violation. They also implement cell re-optimization by means of a recursive depth-first search procedure.

More recent works on MI-FAP-I deal with multi-objective optimization variants of the problem (Aardal et al., 2007). In particular, besides the minimization of the total interference, Laidoui et al. (2018) studied the trade-off between interference and the blocking probability, as a function of the number of frequencies assigned to each cell. Instead, Kiouche et al. (2020) dealt with the problem of simultaneously minimizing also the maximum interference and the number of frequencies used. In both cases, the authors implemented genetic algorithms hybridized by combining elements related

to game theory for Laidoui et al. (2018) and to Artificial Immune Systems for Kiouche et al. (2020).

The MI-FAP-II, also known as *Fixed-Spectrum Frequency-Assignment Problem*, was introduced by Montemanni et al. (2001) as a generalization of the Graph Coloring Problem. The authors propose different lower bounding techniques that are tested on a new dataset, whose main characteristics are discussed in Section 4.4.1. Lower bounds for MI-FAP-II have been further improved in (Montemanni et al., 2004). For the solution of MI-FAP-II, a number of effective Tabu Search algorithms have been proposed by Montemanni et al. (2003), Montemanni and Smith (2010), and Lai and Hao (2015). In particular, Lai and Hao (2015) devise a population based strategy with relinking operators tailored to MI-FAP-II, which were able to create solution paths connecting the two high-quality solutions and generate new promising solutions. The Tabu Search algorithm of Montemanni and Smith (2010) has also been tested on a small subset of the MI-FAP-I dataset.

Segura et al. (2016) developed an evolutionary algorithm with a diversification strategy to avoid premature convergence of the population. This was obtained by converting MI-FAP-II to a multi-objective problem that considers the original objective and, as an auxiliary objective, the contribution of each individual to the diversity. The solution method was evaluated both on MI-FAP-II and on a complex formulation, which arose from two real-world instances coming from the cities of Denver and Seattle (Luna et al., 2007, 2011). In this new formulation, there is no notion of sites and separations involve only transmitters in the same cell (co-cell). Analogously to MI-FAP-I the interference matrix is defined at the cell level, but interferences are not given explicitly and they are computed through a probabilistic model.

Similarly to Lai and Hao, Siddiqi and Sait (2018) proposed a population-based heuristic that employs Tabu Search to drive the exploration of the neighborhood of each solution in the population. The search process is guided by the principles of non-dominated sorting, considering both the interference and the entropy criteria (as a measure of the diversity of individuals of a population).

Finally, Lahsinat et al. (2018) developed a Variable Neighborhood Search (VNS) that explores increasingly large neighborhoods, from the smallest one that changes the frequency of a single transmitter, to the largest that changes simultaneously the frequency of 5 transmitters. In addition, the authors introduced different perturbation schemes for helping the VNS process to escape from local optimum.

Although many works on this topic used neighborhood search, and

specifically Simulated Annealing, to tackle this problem our contribution distinguishes from existing literature mainly in the following two aspects. First of all, differently from the surveyed approaches, we investigate the use of the combination of complex neighborhood structures for solving the problem. Secondly, no previous approach dealt with the MI-FAP-I and MI-FAP-II formulations in a comprehensive way.

4.3 Solution method

Our search method is based on local search, therefore we now introduce, step by step, the *search space* definition, the *initial solution* strategy, the *neighborhood* operators, and the *metaheuristic* that guides the search.

Before proceeding we introduce some notation and terminology that will be useful to illustrate these concepts. We consider the graph in which each single transmitter is taken separately, without their aggregation in cells and sites. This graph is called the *split graph* by Chiarandini and Stützle (2007). Following the graph-coloring terminology, from this point on we will call the transmitters as *nodes*.

We are then given a set of nodes $\mathcal{N} = \{1, \ldots, N\}$ and a set of frequencies $\mathcal{F} = \{1, \ldots, F\}$. We call $\mathbf{S}$ the integer-valued $N \times N$ matrix such that $\mathbf{S}_{n_1,n_2}$ is the required separation between nodes n_1 and n_2. We are also given for each node n a set of frequencies U_n, representing the forbidden frequencies for node n, with $U_n \subset \mathcal{F}$ for all n.

Given these preliminaries we are now able to illustrate the key features of our Multi-Neighborhood Search.

4.3.1 Search space and initial solution

The search space is represented by an integer-valued array φ, so that $\varphi(n)$ is the frequency assigned to node $n \in \mathcal{N}$.

The array φ is complemented by redundant data structures that help us in accelerating the computation of the difference of costs between neighboring solutions (we call them *delta* costs). The main data structure, which has also been used by Chiarandini and Stützle (2007), is an integer-valued matrix $\mathbf{\Gamma}$ that stores, for each pair $\langle n, f \rangle$, the number of conflict violations that would be created by reassigning node n to frequency f in the current state. In addition, we maintain an array of sets Λ that stores, for each node n, the set of nodes that are in conflict (i.e., violated separation) with n in the current state.

For MI-FAP-I, that has real-valued interferences, in order to exploit the faster arithmetic of the integers, we multiply all interference values by a fixed number, suitably high so as not to lose precision (10^8). This is not necessary for MI-FAP-II, for which the values are natively integers.

The initial solution is generated by assigning one node at the time a uniformly-selected random frequency, among those that are not forbidden for that node. That is, separation violations are admitted, but forbidden frequency violations are not. Indeed, forbidden frequency violations are kept outside the search space, as all neighborhoods discussed below do not include moves that reassign a node n to a frequency in U_n.

4.3.2 Multi-neighborhood

The typical neighborhood relation used in FAPs is the replacement of the frequency assigned to one node. We call this neighborhood Change, which is defined as follows:

- Change(C): the move $\mathsf{C}\langle n, f\rangle$ assigns frequency f to node n.
  Preconditions: $\phi(n) \neq f, \quad f \notin U_n$.

The preconditions state that a node must be assigned to a *new* frequency ($\phi(n) \neq f$) and that it cannot be a forbidden one ($f \notin U_n$). Moves that do not satisfy the preconditions are removed from the neighborhood, and thus never drawn.

Similarly to the approach followed by Bellio et al. (2021) for the Examination Timetabling, which has a similar structure, we complement this atomic neighborhood with larger ones that allow us to make more complex movements in one single step. The first one is the so-called Kick move that reallocate two nodes simultaneously, assigning the first one to the frequency of the second one, and the second one to a new frequency.

- Kick(K): the move $\mathsf{K}\langle n_1, n_2, f\rangle$ assigns $\varphi(n_2)$ to n_1 and f to n_2.
  Preconditions: $\mathbf{S}_{n_1,n_2} > 0, \quad \varphi(n_1) \neq \varphi(n_2), \quad \varphi(n_2) \neq f, \quad \varphi(n_2) \notin U_{n_1}, \quad f \notin U_{n_2}$.

The intuition of the Kick neighborhood is to move a node to a potentially favorable frequency even in presence of a conflicting node, given that at the same time the second one is "kicked out".

Fig. 4.1 shows graphically an example of a Kick move, in which the new assignments are shown in light grey.

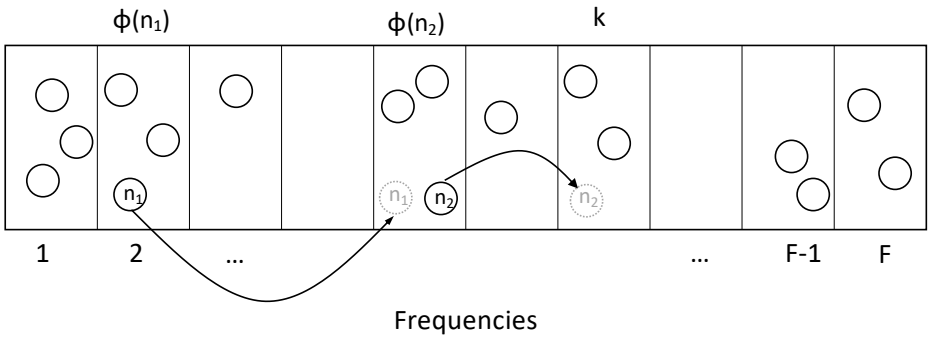


Figure 4.1: A Kick move

The precondition that n_1 and n_2 must have a separation ($\mathbf{S}_{n_1,n_2} > 0$) is meant to restrict Kick moves only to those that are most effective to overcome a cost barrier, with respect to others that could be obtained by a sequence of two cost-independent Change moves. The other preconditions ensure that the move is effective and it does not assign forbidden frequencies.

Notice that f, the new frequency for n_2, could also be equal to $\varphi(n_1)$, resulting in swapping the two assignments. As a consequence, the Kick neighborhood is a superset of the Swap neighborhood, also used in the literature (see Galinier and Hertz, 2006).

The Kick neighborhood is typical for graph coloring problems (see, e.g., González-Velarde and Laguna, 2002). However, FAPs have the peculiarity that conflicts can occur also between nodes assigned to different frequencies. In order to deal with this situation, we propose an extension of the Kick neighborhood, that we call GKick (G for generalized), that moves the first node to a frequency *close* to the second node, and moves the second node away.

- GKick(G): the move $\mathsf{G}\langle n_1, f_1, n_2, f_2\rangle$ assigns f_1 to n_1 and f_2 to n_2.
  Preconditions: $\mathbf{S}_{n_1,n_2} > 0$, $\varphi(n_1) \neq \varphi(n_2)$, $\varphi(n_1) \neq f_1$, $\varphi(n_2) \neq f_2$, $f_1 \notin U_{n_1}$, $f_2 \notin U_{n_2}$, $|f_1 - \varphi(n_2)| < S_{n_1,n_2}$.

The behavior of the GKick neighborhood is analogous to the one of the Kick neighborhood, except that n_1 can be assigned to a frequency f_1 that differs from $\varphi(n_2)$, but is close enough to it that still creates a separation conflict with n_2 (see precondition $|f_1 - \varphi(n_2)| < S_{n_1,n_2}$).

Like for the Kick neighborhood, we can identify here a subset of the neighborhood that we call GSwap in which f_2 is close to $\varphi(n_1)$, in particular

closer than the separation between n_1 and n_2.

As will be shown in Section 4.4.2, the *subneighborhoods* Swap and GSwap play a prominent role in the search. In detail, we will see that it is more effective to bias strongly the random move generation of Kick (resp. GKick) toward Swap (resp. GSwap) moves, rather than to *pure* kicks. For this reason, we include in our neighborhood portfolio a larger one, called 3-Swap, that involves 3 nodes, but makes only swap movements.

- 3-Swap(T): the move $\mathsf{T}\langle n_1, n_2, n_3\rangle$ assigns $\varphi(n_2)$ to n_1, and $\varphi(n_3)$ to n_2, and $\varphi(n_1)$ to n_3.
  Preconditions: $\mathbf{S}_{n_1,n_2} > 0$, $\mathbf{S}_{n_2,n_3} > 0$, $\varphi(n_1) \neq \varphi(n_2)$, $\varphi(n_2) \neq \varphi(n_3)$, $\varphi(n_3) \neq \varphi(n_1)$, $\varphi(n_1) \notin U_{n_2}$, $\varphi(n_2) \notin U_{n_3}$, $\varphi(n_3) \notin U_{n_1}$.

The intuition for introducing the 3-Swap neighborhood is that, on the one hand, we aim at exploring larger neighborhoods, and on the other hand a general 3-node kick (either generalized or not) would be too large, and thus practically ineffective. Therefore, considering the usefulness of the bias toward swap movements for the Kick and GKick neighborhoods mentioned above, we decided that it is wiser to focus only on swaps, and thus the 3-Swap neighborhood seems to be a good trade-off.

Our next neighborhood, called FSwap (F for Frequency), swaps all nodes assigned to one frequency with all assigned to another one.

- FSwap(F): the move $\mathsf{F}\langle f_1, f_2\rangle$ assigns all nodes n such that $\varphi(n) = f_1$ to f_2, and all nodes n such that $\varphi(n) = f_2$ to f_1
  Preconditions: $f_1 \neq f_2$

The FSwap neighborhood is used in graph coloring and its intuition is that the simultaneous movement of all nodes assigned to the same color (frequency in our case) does not increase the number of violations This move however might be less effective in FAPs as, differently from graph coloring, the conflicts come also from nodes in adjacent frequencies. For this reason we define also a partial version of the same move, that swaps a limited number of nodes so that it results less disruptive.

We thus design the neighborhood called PFSwap (P for Partial).

- PFSwap(P): the move $\mathsf{P}\langle f_1, f_2, N_1, N_2\rangle$ assigns all nodes $n_1 \in N_1$ to f_2, and all nodes $n_2 \in N_2$ to f_1
  Preconditions: $f_1 \neq f_2$, for all $n_1 \in N_1$ we have $\varphi(n_1) = f_1$, and for all $n_2 \in N_2$ we have $\varphi(n_2) = f_2$.

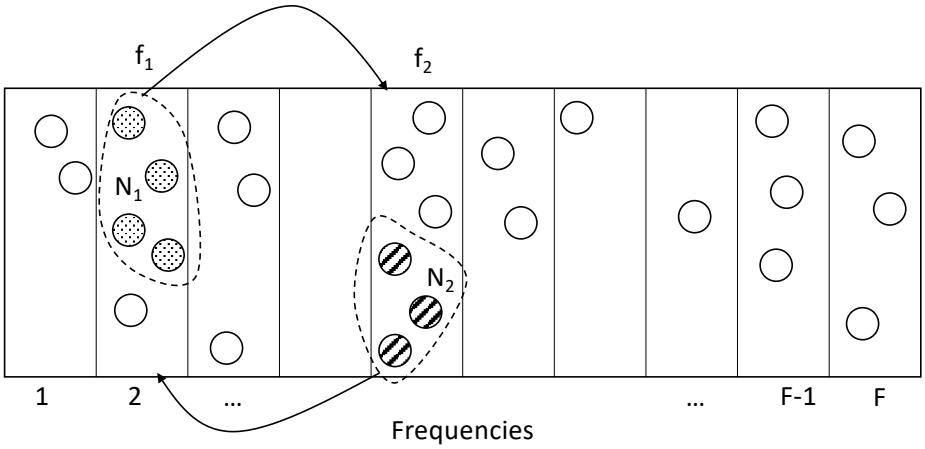


Figure 4.2: A PFSwap move with $k = 7$

The selection of N_1 and N_2 is random, except that the cardinalities are kept equal or differ by one. That is, the selection of the sets is preceded by the selection of a number k, and the number of nodes in N_1 and N_2 is $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$, respectively. The value of k is selected between 3 and a fixed maximum value K_m.

Fig. 4.2 shows graphically an example of a PFSwap move (with $k = 7$).

4.3.3 Metaheuristic and move selection

We employ the Multi-Neighborhood Simulated Annealing described in Chapter 2. Our composite Multi-Neighborhood is Change ∪ Kick ∪ GKick ∪ 3-Swap ∪ FSwap ∪ PFSwap.

The move selection is done in two stages: first we select the atomic neighborhood and then the specific move inside the neighborhood. The first selection is based on the fixed probabilities σ_K, σ_G, σ_T, σ_S, and σ_P, such that at each step neighborhoods Kick, GKick, 3-Swap, FSwap, PFSwap, and Change are selected with probability σ_K, σ_G, σ_T, σ_S, σ_P, and $1 - \sigma_K - \sigma_G - \sigma_T - \sigma_S - \sigma_P$, respectively. Within the single neighborhood, the specific move is selected uniformly, except for Kick and GKick where a move from the Swap (resp. GSwap) sub-neighborhood is selected through an internal bias b_s (resp. b_g) and a pure kick is selected with probability $1 - b_s$ (resp. $1 - b_g$). In this way, b_s and b_g are included in the pool of parameters. In turn, the PFSwap neighborhood is parameterized by the parameter K_m mentioned above, representing the maximum number of nodes involved in the move.

4.3.4 Adaptation to MI-FAP-I and MI-FAP-II

We discuss now how we adapt the local search components introduced above to the specific features of the two versions of the problem. For MI-FAP-I, the presence of many hard constraints due to co-cell, co-site, and handover separations make the problem of finding a feasible solution non-trivial.

Therefore, in order to deal with the feasibility problem properly, but also not to waste time in checking moves that create violations, we make use of a two-stage approach, that performs two independent runs of Simulated Annealing. The first stage starts from a random solution, whereas the second stage starts from the best solution of the first one. Each stage stops according to its own setting of the number of iterations.

In the *Feasibility Stage*, we use only the Change neighborhood and we include also separation violations in the cost function, with a suitably high weight. That is, the cost function that guides the search is the sum of soft constraints (e.g. interference costs) and hard constraints violations. For MI-FAP-I the interference costs occur for same-channel interferences or adjacent channel interferences. On the contrary, hard constraints violations are related to mandatory separations: co-cell frequency separation, co-site frequency separation, frequency separation for cells suffering the handover effect, and ad-hoc separations.

This first stage is rather short in relation to the second one, the *Optimization Stage*, but long enough to obtain a feasible solution. The purpose of the Feasibility Stage is not only to reach feasibility, but also to execute quickly the initial steep descent phase, by using only the Change neighborhood, which is computationally the cheapest.

In the *Optimization Stage*, we use the neighborhood portfolio, and we add the precondition that moves do not introduce violations of hard constraints. As a consequence, for this stage the cost function coincide with the objective function that is to minimize the interference costs.

It turned out that for FSwap and PFSwap it is very rare to find feasible moves, so that most of the time is spent in generating and rejecting moves. For this reason, for MI-FAP-I these two neighborhoods are excluded.

For MI-FAP-II the search method can be simplified in a few ways. First, there are no hard constraints in the strict sense, but a few occurrences of a very high cost. These few high values are considered like all the others, so that the search method proceeds in one single stage. Thus the cost function sums up the cost of transmitter separation violations. Furthermore, there are no forbidden frequencies, so that we don't have to check that a frequency is not available for the node.

The other difference between the two versions is in the computation of the costs: for MI-FAP-II each separation violation must be multiplied by its weight, whereas for MI-FAP-I interference depends on the distance (same or adjacent channel).

A peculiarity of Frequency Assignment problems, that is shared by MI-FAP-I and MI-FAP-II, is that certain nodes are *indistinguishable*, in the sense that they share exactly the same separations and interference levels. For MI-FAP-I this is made explicit by the fact that they are members of the same cell and of the same type (control or traffic), for MI-FAP-II indistinguishable pairs are detected by preprocessing the input data. To improve efficiency, we remove moves that involve two indistinguishable nodes, as they would not change the structure and the cost of the solution.

4.4 Experimental results

The software was implemented in C++ and compiled using g++ (v. 9.3) in `-O3` mode. The experiments were run on AMD Ryzen Threadripper PRO 3975WX 32-Cores (3.50 GHz) with Ubuntu Linux 20.4. One single core was dedicated to each experiment.

4.4.1 Benchmarks

Both formulations are equipped with a specific dataset that has been used in previous works. We refer to Eisenblätter and Koster (2000) and Montemanni (2001), respectively, for an accurate description of the origin of these instances.

In order to save time, we decided to identify and remove the "easy" instances. We classify as easy those instances in which the same scores are obtained consistently by all configurations of our technique and by the most effective previous works.

The features of the non-easy instances are summarized in Tables 4.1 and 4.2, for MI-FAP-I and MI-FAP-II respectively. For MI-FAP-I we consider the number of sites (S), the number of cells (C), the number of nodes/transmitters (N), the number of frequencies (F), and the average number of available frequencies per transmitter (AF). We consider the split graph corresponding to the separations and we report its density (SD) and the average separation value (SV). Finally, we report the density of the interference split graph (ID).

For MI-FAP-II we consider the number of nodes/transmitters (N), the list of numbers of frequencies (Fs), the density of the interference graph (ID), the

average separation (IS), and the average cost (P). For this version, the same instance is used with different number of frequencies, therefore we report here the list of them (Fs) rather than a single value (F). From the average costs we exclude the artificial high value used to state the hard separations. The number of hard separations is reported in the last column (H).

Notice that MI-FAP-I instances are generally much larger than those of MI-FAP-II in terms of number of nodes. For MI-FAP-I, we also notice that only four instances, namely siemens1, siemens2, siemens3 and swisscom, have forbidden frequencies, shown by the fact that the value of the column AF is smaller than the value in the column F. In particular, siemens1 and siemens2 have only globally forbidden frequencies, whereas siemens3 and swisscom have also locally forbidden ones. For swisscom the AF value is particularly low, and only for this instance it is particularly difficult to find a feasible solution.

Regarding MI-FAP-II, we notice that, based on the average separation cost (P), the dataset can be split on three distinct groups of instances. In detail, there is a group that has all costs equal to one, a second group with medium values (on the order of tens), and a final one with larger costs (on the order of thousands). This partition reflects, with a few exceptions, the different generation procedures: the first set is obtained from existing minimum span problems by limiting the number of available frequencies; the second set is composed of random scenarios generated using a basic graph generator, and the last set is obtained by adapting some fixed spectrum GSM problems to MI-FAP-II. As the cost values have a significant impact on the SA behavior, as described in the next section, we consequently perform separate tuning for these three groups.

Other datasets for MI-FAP have been proposed in the literature, like the Philadelphia and CALMA ones (see Anderson, 1973; Aardal et al., 2002). These cases however are rather simple to solve (most of the instances have been solved to optimality), so that we decided to skip them. On the contrary, the two instances Denver and Seattle proposed by Luna et al. (2007) are supposed to be difficult, but unfortunately they refer to a different version of the problem.

We translated MI-FAP-I instances from their original file format to a novel JSON-based one. The original format is rather complex to parse, so that we believe that this new one could foster the dissemination of these instances, which in fact has been quite limited in the recent times. On the contrary, for MI-FAP-II, the original format is extremely simple, so that we kept it as is.

Table 4.1: Features of the instances for MI-FAP-I.

Instance	S	C	N	F	AF	SD	SV	ID
bradford_nt-1-eplus	649	1886	1971	75	75.0	0.0041	2.01	0.1305
bradford_nt-10-eplus	649	1886	4145	75	75.0	0.0045	1.90	0.1305
bradford_nt-10-free	649	1886	4145	75	75.0	0.0044	1.90	0.0525
bradford_nt-10-race	649	1886	4145	75	75.0	0.0044	1.90	0.0406
bradford_nt-2-eplus	649	1886	2214	75	75.0	0.0042	2.02	0.1310
bradford_nt-4-eplus	649	1886	2775	75	75.0	0.0043	1.99	0.1304
bradford-0-eplus	649	1886	1886	75	75.0	0.0041	2.00	0.1319
bradford-1-eplus	645	1878	2947	75	75.0	0.0050	2.13	0.1286
bradford-1-free	645	1878	2947	75	75.0	0.0049	2.13	0.0519
bradford-1-race	645	1878	2947	75	75.0	0.0049	2.13	0.0392
bradford-10-eplus	644	1876	4871	75	75.0	0.0052	2.02	0.1314
bradford-10-free	644	1876	4871	75	75.0	0.0052	2.02	0.0532
bradford-10-race	644	1876	4871	75	75.0	0.0052	2.03	0.0395
bradford-2-eplus	644	1876	3406	75	75.0	0.0051	2.12	0.1295
bradford-2-free	644	1876	3406	75	75.0	0.0051	2.12	0.0524
bradford-2-race	644	1876	3406	75	75.0	0.0051	2.12	0.0389
bradford-4-eplus	649	1886	3996	75	75.0	0.0051	2.08	0.1292
bradford-4-free	649	1886	3996	75	75.0	0.0051	2.09	0.0526
bradford-4-race	649	1886	3996	75	75.0	0.0051	2.09	0.0385
K	92	264	267	50	50.0	0.0297	2.00	0.5382
siemens1	179	506	930	75	43.0	0.0140	2.07	0.0764
siemens2	86	254	977	83	76.0	0.0373	2.08	0.4550
siemens3	366	894	1623	55	51.2	0.0175	2.03	0.0743
siemens4	276	760	2785	39	39.0	0.0072	2.14	0.0978
swisscom	87	148	310	68	29.0	0.0832	1.53	0.0433

Table 4.2: Features of the instances for MI-FAP-II.

Instance	N	Fs	ID	IS	P	H
AC-95-17	95	15	0.51	1.15	1.00	0
GSM-93	93	9/13	0.25	1.28	1.00	0
GSM-246	246	21/31	0.25	1.32	1.00	0
Test95	95	36	0.27	2.37	1.00	0
Test282	282	61/71/81	0.26	2.38	1.00	0
P06-3	153	31	0.79	1.59	1.00	0
P06-5	88	11	0.79	1.58	1.00	0
P06b-3	153	31	0.79	1.39	1.00	0
GSM2-184	184	39	0.40	1.20	1670.23	609
GSM2-227	227	29/39/49	0.39	1.18	1746.91	918
GSM2-272	272	34/39/49	0.39	1.16	1721.07	1155
1-1-50-75-30-2-50	75	5/10/11/12	0.30	1.26	10.81	0
1-2-50-75-30-4-50	75	9/11	0.30	1.62	11.09	0
1-3-50-75-30-0-50	75	7	0.30	1.00	10.97	0
1-4-50-75-30-2-1	75	6/10	0.30	1.25	1.00	0
1-5-50-75-30-2-100	75	10/12	0.30	1.26	21.35	0
1-6-50-75-30-0-1000	75	10/13	0.30	1.00	2068.48	0

4.4.2 Parameter tuning

The tuning procedure was performed using the tool JSON2RUN (Urli, 2013), which uses configurations generated according to Hammersley and Handscomb (1964), known as the Hammersley point set. JSON2RUN uses the F-Race procedure (Birattari et al., 2010) for selecting the best configuration, which is based on the Friedman and Wilcoxon statistical tests for removing inferior configurations as soon as possible.

The total number of parameters is quite large, hence the parameter tuning proceeds in phases, assuming that the interaction of the parameters involved in the different phases is minimal and can be neglected. In each phase, the parameters belonging to a subsequent phase are set to values given from preliminary experiments.

The winning configuration for MI-FAP-I is shown in Table 4.3, obtained with $\mathcal{I} = 3 \cdot 10^8$ corresponding to a running time of approximately 1500 seconds per run.

Notice the high values of the temperatures, which are due to the fact that the interference values are integers, obtained by multiplying the actual values by 10^8 (for full precision). Notice also the high values of the two bias parameters b_s and b_g, which show that the most useful kick moves are indeed swap ones.

The tuning procedure for MI-FAP-II works along the same tracks, except that there is no Feasibility Stage and the tuning is done separately for the three groups of instances, due to the different cost values, which influence the corresponding temperature ranges. Furthermore, there is an extra parameter K_m which is the maximum length of a PFSwap move, which is not in Table 4.3 as PFSwap is not used for MI-FAP-I. The best value found for K_m is 4.

4.4.3 Comparison results for MI-FAP-I

In our experience, for this problem the results improve consistently with the running time, without any sort of "plateau effect". As a consequence, the comparison should take into account the running times and also the CPU speed. Unfortunately though, a comparison of different CPUs in different years is rather impractical. In addition, Mannino et al. (2007), which hold most of the best results so far, granted an extremely long running time to their experiments (i.e., up to 128 hours per run, depending on the instance).

Therefore, as a fair comparison is not possible and the running times of Mannino et al. (2007) are impractical also for future comparisons, we decided for this version of the problem to grant our experiments a fixed

Table 4.3: Parameter tuning for MI-FAP-I.

Name	Description	Tuning Phase	Initial Range	Value
	Feasibility Stage			
T_0	Start temperature	1	[500000, 2500000]	1314815
T_f	Final temperature	1	[1000, 10000]	9280
α	Cooling rate	1	[0.98, 0.999]	0.995
ρ	Accepted moves ratio	1	[0.03, 0.15]	0.076
	Optimization Stage			
T_0	Start temperature	2	[300000, 1000000]	697531
T_f	Final temperature	2	[1000, 10000]	8632
α	Cooling rate	2	[0.98, 0.999]	0.985
ρ	Accepted moves ratio	2	[0.05, 0.2]	0.112
b_s	Bias toward swap moves	3	[0.0, 1.0]	0.906
b_g	Bias toward generalized swap moves	3	[0.0, 1.0]	0.906
σ_{K}	Probability of Kick moves	4	[0.0, 0.4]	0.216
σ_{G}	Probability of GKick moves	4	[0.0, 0.4]	0.042
σ_{T}	Probability of 3-Swap moves	4	[0.0, 0.2]	0.009

number of iterations, specifically equal to $\mathcal{I} = 3 \cdot 10^9$. The results for 10 runs in comparison with the best in the literature are shown in Table 4.4. Beckmann and Killat (1999) and Hellebrandt and Heller (2000) do not report the running times. Montemanni et al. (2003) write that the experiments run for "several days". Montemanni and Smith (2010) set a timeout of 2h for all instances.

We can see that we outperformed all previous results on most of the instances, considering both the best and the average values. Only in six cases our average results are slightly inferior to the best ones, which however are obtained with much longer running time (on an older CPU, tough). In addition, we improve the best known solutions for 23 out of 25 instances.

4.4.4 Comparison results for MI-FAP-II

Table 4.5 shows the results for 30 runs with timeout 2400s, which has been set also by the other authors. For the path relinking approach by Lai and Hao (2015), we report the results obtained by both the random path relinking operator (denoted with rPR) and the randomized and mixed relinking operator (denoted with mrPR).

Table 4.4: Computational results for MI-FAP-I on COST 259 instances.

	Beckmann 1999	Hellebrandt 2000	Montemanni 2003	Montemanni 2010		Mannino 2007		SA		
instance	best	best	best	best	avg	best	t[h]	best	avg	t[h]
bradford_nt-1-eplus	1.04	**0.86**				**0.86**	22	0.871	0.951	2.1
bradford_nt-10-eplus	148.12	146.12				144.94	19	**142.746**	143.719	3.7
bradford_nt-10-free	8.63	5.863				5.42	14	**4.945**	5.213	3.3
bradford_nt-10-race	1.73	1.074				1.09	14	**1.035**	1.077	4.2
bradford_nt-2-eplus	3.79	3.168				3.20	24	**3.152**	3.372	2.3
bradford_nt-4-eplus	19	17.728				17.72	21	**17.209**	17.682	2.7
bradford-0-eplus	0.8					0.60	64	**0.597**	0.641	2.0
bradford-1-eplus	33.99					33.80	64	**32.381**	32.735	3.2
bradford-1-free	0.16					**0.12**	30	0.164	0.172	3.2
bradford-1-race	0.03					0.01	26	**0.009**	0.011	4.0
bradford-10-eplus	400					395.50	128	**387.206**	388.573	5.9
bradford-10-free	117.8					113.70	63	**104.612**	105.997	4.7
bradford-10-race	30.22					27.38	46	**23.758**	24.072	5.3
bradford-2-eplus	80.03					79.38	75	**76.674**	76.984	3.9
bradford-2-free	2.95					2.69	37	**2.275**	2.365	3.6
bradford-2-race	0.42					0.32	39	**0.201**	0.222	4.3
bradford-4-eplus	167.7					167.00	90	**161.325**	162.894	4.5
bradford-4-free	22.09					20.00	46	**17.883**	18.348	3.9
bradford-4-race	3.04					2.93	36	**2.174**	2.277	4.7
K		0.45	0.447	0.4647	0.4886			**0.415**	0.434	0.4
siemens1	2.78	2.301		2.7642	2.8492	2.20	5	**1.970**	2.035	0.9
siemens2	15.46	14.751	14.275	14.936	15.0578	14.27	9	**14.005**	14.156	2.7
siemens3	6.75	5.259	5.186	6.6496	6.7358	5.13	15	**4.852**	4.971	3.1
siemens4	89.15	80.967	81.876	110.9725	112.482	77.25	18	**76.298**	77.014	5.6
swisscom	27.36		27.211					**27.027**	29.444	1.3

Table 4.5: Computational results for MI-FAP-II with a time-limit of 2400 secs.

		Montemanni 2003	Montemanni 2010		Lai 2015				Lahsinat 2018		SA	
					rPR		mrPR					
Instance	F	best	best	avg	best	avg	best	avg	best	avg	best	avg
AC-95-17	15	33	33	**33.0**	33	**33.0**	33	**33.0**			33	33.1
GSM-93	9	32	32	**32.2**	32	**32.2**	32	**32.2**	33	34.18	32	33.2
GSM-93	13	7	7	**7.0**	7	**7.0**	7	**7.0**	8	8.60	7	**7.0**
GSM-246	21	79	79	80.2	79	80.6	78	79.0	83	84.78	77	**78.9**
GSM-246	31	25	25	26.1	26	26.1	24	25.1			24	**24.7**
Test95	36	12	8	**8.0**	8	**8.0**	8	**8.0**	8	**8.00**	8	**8.0**
Test282	61	51	51	**53.2**	56	56.8	56	57.1			51	54.3
Test282	71	27	27	29.3	29	30.5	29	30.6			27	**28.9**
Test282	81		10	11.9	9	10.9	10	11.5			9	**10.5**
P06-5	11	133	133	**133.0**	133	**133.0**	133	**133.0**	137	137.26	133	**133.0**
P06-3	31	115	115	**115.0**	115	**115.0**	115	**115.0**	115	115.10	115	**115.0**
P06b-3	31	112	112	**112.0**	112	**112.0**	112	**112.0**	112	117.00	112	**112.0**
GSM2-184	39	5521	5447	5598.8	5258	**5270.8**	5250	5276.9	5898	6180.71	5250	5279.9
GSM2-227	29		61586	66510.0	57790	59555.4	58834	59907.7	67586	68721.00	56122	**57932.6**
GSM2-227	39	10979	10550	10897.7	8656	**9022.4**	8760	9329.7			8702	9087.4
GSM2-227	49	2459	2459	2613.1	1998	**1998.0**	1998	2009.4			1998	2019.0
GSM2-272	34		56128	58691.4	53254	55954.2	54085	56916.3	65150	67888.30	51579	**53118.9**
GSM2-272	39	27416	27416	28488.2	27503	28299.7	28074	28880.4			26479	**27165.1**
GSM2-272	49	7785	7785	7946.7	7185	7265.2	7107	7252.5			7075	**7169.4**
1-1-50-75-30-2-50	5		1242	1253.9	1242	**1242.0**	1242	**1242.0**	1257	1268.44	1242	**1242.0**
1-1-50-75-30-2-50	10		97	103.8	96	**96.0**	96	**96.0**			96	96.2
1-1-50-75-30-2-50	11		59	66.1	55	**55.0**	55	**55.0**			55	56.2
1-1-50-75-30-2-50	12		36	38.7	32	**32.0**	32	**32.0**			32	32.6
1-2-50-75-30-4-50	9		671	680.6	665	**665.0**	665	**665.0**	670	674.18	665	**665.0**
1-2-50-75-30-4-50	11		317	325.0	313	**313.0**	313	**313.0**			313	313.6
1-3-50-75-30-0-50	7		194	196.5	194	**194.0**	194	**194.0**	196	196.76	194	**194.0**
1-4-50-75-30-2-1	6		70	70.9	70	**70.0**	70	**70.0**	71	74.20	70	**70.0**
1-4-50-75-30-2-1	10		19	**19.0**	19	**19.0**	19	**19.0**			19	**19.0**
1-5-50-75-30-2-100	10		176	183.8	168	**168.0**	168	**168.0**			168	173.6
1-5-50-75-30-2-100	12		63	69.3	57	**57.0**	57	**57.0**			57	57.7
1-6-50-75-30-0-1000	10		6840	7064.3	6777	**6777.0**	6777	**6777.0**			6777	**6777.0**
1-6-50-75-30-0-1000	13		1207	1365.2	1190	**1190.0**	1190	**1190.0**			1190	**1190.0**

We can see that for all instances excluding instance GSM2-227 we reach the best known result, whereas the average results are in some cases worse than the previous ones, mainly by Lai and Hao. For the GSM2 instances, we have the best average results for 4 out of 7 instances.

In Table 4.6, we compare our results with those obtained by the hybrid genetic algorithm (HGA) of Siddiqi and Sait (2018) with a common time-limit of maximum 2 hours. For SA, the table reports the best and average values of 10 runs. It can be noticed that for all instances with a separation cost equal to one or to medium values, the performances of the two methods are almost equivalent with 14 ties, four instances for which the results of SA are better than those of HGA and five for which SA is worse. Conversely, for the last family, which comprises the GSM2* instances and the instance 1-6-50-75-30-0-1000 and that is characterized by large separation costs, HGA exhibits superior results, being better, equal and worse than SA in five, two and two cases, respectively.

Finally, in Table 4.7 we present comparative results on a subset of the MI-FAP-II dataset composed by the most challenging instances for a time-limit of 48 hours. The first column reports some lower bounds (LB) computed by Montemanni et al. (2004); for the evolutionary algorithm (EA) developed by Segura et al. (2016), the table shows average and best values of 30 runs, for SA those of 5 runs. It can be noticed that SA outperforms EA in nine out of 13 instances, founding four new best known solutions. These last cases are marked with an $*$ in the column corresponding to the best values obtained by SA. We want to remark that the EA was specifically designed to deal with long-term executions, and it has not been tested on shorter running times. Indeed, the authors themselves claim that with short time "high-quality results could not be obtained".

4.5 Discussion

Our solver works reasonably well for both formulations, though the results are somewhat better for MI-FAP-I than for MI-FAP-II. This shows that it is particularly competitive for large instances and the more complex structure.

We see that the approach of Montemanni and Smith (2010), which has good results for MI-FAP-II, works less effectively for the few instances of MI-FAP-I upon which it has been tested. The other approaches that performed well on MI-FAP-II have not been applied to MI-FAP-I, therefore we cannot make any conclusions on their potential performance on this version.

Table 4.6: Computational results for MI-FAP-II with a time-limit of 2 hours.

		HGA		SA	
Instance	F	best	avg	best	avg
AC-95-17	15	33	**33.0**	33	**33.0**
GSM-93	9	32	**32.0**	32	33.2
GSM-93	13	7	**7.0**	7	**7.0**
GSM-246	21	78	79.2	78	**78.6**
GSM-246	31	24	24.8	24	**24.3**
Test95	36	8	**8.0**	8	**8.0**
Test282	61	52	53.8	51	**53.5**
Test282	71	27	**27.4**	26	27.5
Test282	81	8	**8.3**	8	9.5
P06-5	11	133	**133.0**	133	**133.0**
P06-3	31	115	**115.0**	115	**115.0**
P06b-3	31	112	**112.0**	112	**112.0**
GSM2-184	39	5250	5265.0	5258	**5264.4**
GSM2-227	29	55513	**56789.0**	56464	57474.7
GSM2-227	39	8520	**8700.3**	8762	8911.3
GSM2-227	49	1998	**1998.0**	1998	2002.0
GSM2-272	34	51493	**52354.5**	51877	52907.1
GSM2-272	39	25932	**26685.0**	26198	26766.4
GSM2-272	49	7056	7129.4	7017	**7089.0**
1-1-50-75-30-2-50	5	1242	**1242.0**	1242	**1242.0**
1-1-50-75-30-2-50	10	96	**96.0**	96	**96.0**
1-1-50-75-30-2-50	11	55	**55.0**	55	**55.0**
1-1-50-75-30-2-50	12	32	**32.8**	32	**32.0**
1-2-50-75-30-4-50	9	665	**665.0**	665	**665.0**
1-2-50-75-30-4-50	11	313	**313.0**	313	**313.0**
1-3-50-75-30-0-50	7	194	**194.0**	194	**194.0**
1-4-50-75-30-2-1	6	70	**70.0**	70	**70.0**
1-4-50-75-30-2-1	10	19	**19.0**	19	**19.0**
1-5-50-75-30-2-100	10	168	**168.0**	168	169.2
1-5-50-75-30-2-100	12	53	**56.5**	57	57.0
1-6-50-75-30-0-1000	10	6777	**6777.0**	6777	**6777.0**
1-6-50-75-30-0-1000	13	1190	**1190.0**	1190	**1190.0**

Table 4.7: Computational results for MI-FAP-II with a time-limit of 48 hours.

		LB	EA		SA	
Instance	F	value	best	avg	best	avg
GSM-246	21	50	77	79.2	77	**77.8**
GSM-246	31	16	25	26.2	*23	**23.6**
Test282	61	21	53	54.7	*50	**50.6**
Test282	71	6	27	28.7	*25	**25.4**
Test282	81		8	9.9	*7	**7.8**
GSM2-184	39	4856	5250	5251.6	5250	**5251.2**
GSM2-227	29		55339	**56349.0**	55796	56447.4
GSM2-227	39	7445	8283	**8567.0**	8467	8580.4
GSM2-227	49	1998	1998	**1998.0**	1998	**1998.0**
GSM2-272	34		50940	51757.0	50959	**51565.0**
GSM2-272	39	16144	25542	26099.6	25780	**25923.4**
GSM2-272	49	6310	6957	7096.6	6978	**7012.6**
1-5-50-75-30-2-100	10	94	168	**168.0**	168	**168.0**

In addition, the MNSA method obtains very competitive results for both short and long executions, proving that it is flexible to different timeout. In particular, with the long runs it improved many best known results.

4.5.1 Larger neighborhoods

It would be worth discussing whether larger neighborhoods could contribute to improve the results. For example, we could consider the X-Swap neighborhood (with $X > 3$), i.e., the generalization of 3-Swap. However, the tuning experiments showed that the contribution of the 3-Swap neighborhood in the overall best configuration is rather limited ($\sigma_T = 0.009$). They also showed (not reported in the paper) that the configurations with even lower values of σ_T (even $\sigma_T = 0.0$) do not have a significant loss of performance. In addition, X-Swap would result in a more complex neighborhood structure with a less efficient evaluation of the delta costs. For these reasons, we decided not to investigate further in the X-Swap direction.

4.5.2 Instance-based tuning

Some additional insights about the results come from the analysis of the ratio between the time to find the best solution and the total elapsed time. In most instances this ratio is close to 1, showing that the best solution is found toward the end of the search. This is a positive behavior that shows that no time is wasted during the search.

There are however a few instances in which this ratio is constantly much lower than 1. In particular, for one specific instance, namely swisscom, this ratio turned out to be extremely low (around 0.01). This is a very peculiar and constrained instance, in which finding a feasible solution is much more difficult than in all the others. This behavior rises the question whether the parameter values coming for the general tuning are suitable for this "outlier".

Additional experiments on this instance alone proved that an ad-hoc tuning yields to different values (in particular a much higher value for T_0), which would result in much better scores. Specifically, we obtain an average cost of 25.99 and a best one of 23.478, compared to 29.444 and 27.027 of Table 4.4, respectively.

This is however the result of an "overtuning", which is methodologically unacceptable, as the tuning procedure is expected to prepare the method for generic unforeseen instances. Otherwise, the tuning procedure should be considered as part of the solution of the instance and its time should be included in the solution time.

Nonetheless, this situation might pave the way for a feature-based tuning that relates the parameters of the search method to the features of the instance. This however would require a much larger dataset of instances, and thus will be subject of future work.

4.6 Conclusions

We have proposed a multi-neighborhood Simulated Annealing approach for the MI-FAP problem. The solver has been designed to deal with two versions of the problem (with some adaptations), that we called MI-FAP-I and MI-FAP-II. Our solver proved to be effective and robust on both formulations, on many diverse instances, and with different time-limits, and compares favorably with previous results.

Most of the recent work focused on the MI-FAP-II version, probably due to the higher structural complexity of the MI-FAP-I formulation, and maybe also to the larger size of its instances. Another reason for the lack of recent "success" of the MI-FAP-I formulation might be its cumbersome file format.

To this regard, we have translated all instances to a novel JSON format, with the expectation that this "restyling" might bring it back on tracks for future comparisons.

To this aim, the publication of instances and solutions, along with the online validator, might also attract new research on this interesting problem.

Chapter 5

Sports Timetabling

Sports timetabling is an active research field, mainly due to the commercial interest in the maximization of fan attendance, in person or remotely, to sport events. Among the various possible structures for sport competitions, the round-robin tournament, where each team plays against each other, is the most frequently used for most team sports.

Many variants of the round-robin tournament problem have been discussed in the literature. We consider here the version proposed for the International Timetabling Competition ITC2021 (Van Bulck and Goossens, 2023b): a double round-robin tournament (all teams play with each team twice), which takes into account a very rich set of constraints and objectives collected from real-world cases.

All versions of this problem have in common the fact of being generally difficult to solve in practice. In fact, it is often hard to find optimal (or near-optimal) solutions already for instances of relatively small sizes, i.e., 16-20 teams, which is indeed the typical size of national championships.

As mentioned above, the ITC2021 problem considers a large set of constraints and objectives, also known as *hard* and *soft* constraints, respectively. This formulation has the peculiarity that every single specific constraint can be stated as either hard or soft. Another characteristic of the ITC2021 formulation is that it has abandoned the classical *mirrored* structure in which the second leg is identical to the first one, with home and away positions swapped. That is, the structure of ITC2021 instances is either completely free or *phased*. The latter imposes that each team meets all other teams in each leg, but not necessarily in the same order.

In this chapter, we describe the Multi-Neighborhood Simulated Annealing employed in our participation in the ITC2021. It is a three-stage

approach, that uses a portfolio of six different neighborhood structures. Five of them are classical ones, already proposed in the literature, whereas the sixth one, named PartialSwapTeamsPhased, is novel neighborhood that we specifically designed to deal with phased instances. Simulated Annealing has been used also by other authors for sports timetabling with good results, suggesting that it is particularly suitable for this type of problems (see Section 5.1 on Related Work).

Our solver has many parameters, and it has been tuned using the F-RACE procedure (Birattari et al., 2010), upon a set of experimental configurations designed using the Hammersley point set (Hammersley and Handscomb, 1964).

We also propose an Integer Linear Programming (ILP) model for the problem. We implemented it in CPLEX but, unfortunately, it was able to solve systematically only small artificially generated instances, and it did not produce significant results on the instances of the competition even after long running times.

5.1 Related work

Interest in Sports timetabling dates back to the 70s. Initial research by Gelling (1973), Russell (1980), Wallis (1983), and de Werra et al. (1990) focused on the relationship between 1-factorizations of a complete graph and the Sports timetabling Problem. In sports timetabling, 1-factorizations take the name of *patterns* and de Werra (1981) proposed an easy way to generate a 1-factorization, that has been named *canonical pattern*. Nevertheless, Rosa and Wallis (1982) and Dinitz et al. (1994) warned about the complexity in the generation of non-isomorphic 1-factorizations. Due to its complexity, applications of metaheuristics to the Sports timetabling Problem date back to the 90s, with contributions from Costa (1995), Della Croce et al. (1999) and Hamiez and Hao (2000). In the 2000s Ribeiro and Urrutia (2007), Anagnostopoulos et al. (2006), and Di Gaspero and Schaerf (2007) proposed a set of new neighborhoods for local-search-based metaheuristics. They have been employed either with Tabu Search or Simulated Annealing and were particularly effective for the solution of the Traveling Tournament Problem (TTP), proposed by Easton et al. (2001).

In the last decade, Lewis and Thompson (2011), Costa et al. (2012), and Januario and Urrutia (2016) worked on further heuristics and new neighborhoods for the solution of the Sports timetabling Problem. More recently, Van Bulck et al. (2020) introduced a unified data format for the

round-robin sports timetabling, named RobinX, that synthesize 18 different constraints belonging to five different constraint groups, and they published a large set of instances in the proposed format. The RobinX format is employed in the Sports timetabling Competition ITC2021 (Van Bulck and Goossens, 2023b).

More complete bibliographic revisions for sports timetabling can be found in Rasmussen and Trick (2008), and Kendall et al. (2010). Finally, an up-to-date bibliography is also available online and maintained by Knust (2010).

5.2 Problem formulation

We introduce here the ITC2021 problem through its Integer Linear Programming (ILP) model, and we refer to Van Bulck and Goossens (2023b) for a comprehensive presentation.

Let n be an even number and $\mathcal{T} = \{1, \dots, n\}$ be the set of teams. In a double round-robin tournament, each team $i \in \mathcal{T}$ plays a game against each other team $j \in \mathcal{T}$, $j \neq i$ twice, once at home and once away. We identify the home and away games of team i against team j, respectively, with the pairs (i, j) and (j, i). Hence, the set of games that have to be scheduled in the league is $\mathcal{G} = \{(i, j) \in \mathcal{T} \times \mathcal{T} : i \neq j\}$. In addition, in a time-constrained tournament the number of rounds available to schedule the games of $\mathcal{G}$ has to be minimal. Then, $\mathcal{R} = \{1, \dots, 2(n-1)\}$ is the set of the available rounds in a double round-robin tournament and, at every round $r \in \mathcal{R}$, each team plays exactly once, either at home or away. A timetable is an assignment of exactly one round of $\mathcal{R}$ to each game in $\mathcal{G}$. We say that a timetable is *phased* if the season is split in two legs, and each team plays against all the other teams exactly once in each leg: team i and j cannot play both their mutual games (i, j) and (j, i) in the same leg. In phased timetables, the first leg occurs in rounds $1, \dots, |\mathcal{R}|/2$, whereas the second one in rounds $|\mathcal{R}|/2 + 1, \dots, |\mathcal{R}|$.

Sports timetables usually consider several additional constraints. Specifically, we consider five groups of constraints. *Capacity constraints* regulate the number of home games, away games or games that a team or a subset of teams can play in a given subset of rounds. *Game constraints* fix or forbid specific assignments of games to rounds. *Break constraints* are used to limit the number of *breaks*, that is the number of consecutive home or away games for a team. Breaks are mostly undesired in a fair timetable. *Fairness constraints* limit the difference of home games played by two teams after each

round. Finally, *Separation constraints* ensure that the mutual games of two teams are separated by a given number of rounds. We call $\mathcal{C}$ the set of these constraints. Set $\mathcal{C}$ contains *hard* and *soft* constraints: the former express fundamental properties of the timetable and must be satisfied, whereas the latter express preferences and can be violated. We denote by $\mathcal{C}_{hard}$ and $\mathcal{C}_{soft}$ the subsets of $\mathcal{C}$ containing, respectively, the hard and soft constraints. For each soft constraint $c \in \mathcal{C}_{soft}$, we denote by w_c the weight associated to its violation.

For each game $(i, j) \in \mathcal{G}$ and each round $r \in \mathcal{R}$, we introduce a binary variable x_{ijr} defined as follows

$$x_{ijr} = \begin{cases} 1 & \text{if game } (i,j) \text{ is played in round } r \\ 0 & \text{otherwise.} \end{cases}$$

For each soft constraint $c \in \mathcal{C}_{soft}$, we include a non-negative continuous variable d_c representing the deviation triggered if the constraint is violated.

The model, denoted by $\mathcal{M}$, reads as follows.

$$min \sum_{c \in \mathcal{C}_{soft}} w_c d_c \tag{5.1}$$

$$\sum_{r \in \mathcal{R}} x_{ijr} = 1 \qquad \forall (i,j) \in \mathcal{G} \tag{5.2}$$

$$\sum_{j \in \mathcal{T}, j \neq i} x_{ijr} + x_{jir} \leq 1 \qquad \forall i \in \mathcal{T}, \forall r \in \mathcal{R} \tag{5.3}$$

$$\sum_{r \in \mathcal{R}, r \leq |\mathcal{R}|/2} x_{ijr} + x_{jir} = 1 \qquad \forall (i,j) \in \mathcal{G}, i < j. \tag{5.4}$$

Objective function (5.1) minimizes the weighted violation of the soft constraints. Constraints (5.2) and (5.3) define a timetable for the games in $\mathcal{G}$ in the rounds of $\mathcal{R}$. Specifically, Constraints (5.2) impose that every game is played, i.e. it is assigned to exactly one round, and Constraints (5.3) ensure that each team plays at most one time per round. Finally, Constraints (5.4) guarantee that the timetable is phased, if required.

In the following, we list the constraint types considered in set $\mathcal{C}$. We first discuss the hard version of these constraints. Some additional notation, such as subsets of teams or rounds and parameters, may be required for each constraint $c \in \mathcal{C}$. For example, if constraints c is identified by a team $i \in \mathcal{T}$, we denote by $\mathcal{T}(i)$ and/or $\mathcal{R}(i)$, respectively, the subsets of teams and

rounds considered by the constraint itself: the dependency on c is dropped to lighten the notation. Furthermore, we explicit the correspondence between the constraints in set $\mathcal{C}$ and those considered in Van Bulck and Goossens (2023b) to avoid ambiguities.

- **Capacity Constraints (CA).**

$$\underline{k}(i) \leq \sum_{j \in \mathcal{T}(i), j \neq i} \sum_{r \in \mathcal{R}(i)} x_{ijr} \leq \bar{k}(i) \qquad \forall i \in \mathcal{T} \tag{5.5}$$

$$\underline{k}(i) \leq \sum_{j \in \mathcal{T}(i), j \neq i} \sum_{r'=r}^{r+\bar{r}(i)} x_{ijr'} \leq \bar{k}(i) \quad \forall i \in \mathcal{T}', \forall r = 1, \ldots, |\mathcal{R}| - \bar{r}(i) + 1 \tag{5.6}$$

$$\underline{k} \leq \sum_{i \in \mathcal{T}'} \sum_{j \in \mathcal{T}'', j \neq i} \sum_{r \in \mathcal{R}'} x_{ijr} \leq \bar{k} \qquad \forall \mathcal{T}', \mathcal{T}'' \subseteq \mathcal{T}, \forall \mathcal{R}' \subseteq \mathcal{R}. \tag{5.7}$$

  Constraints (5.5) (CA1 and CA2 in Van Bulck and Goossens, 2023b) impose that team i plays at least $\underline{k}(i)$ and at most $\bar{k}(i)$ home games against the teams in subset $\mathcal{T}(i) \subseteq \mathcal{T}$ in the rounds of set $\mathcal{R}(i) \subseteq \mathcal{R}$. These constraints can be used to model the so-called *place constraints* that forbid a team to play at home in a given round and the so-called *top team and bottom team constraints* which avoid bottom teams to play all the initial games against top teams. Then, Constraints (5.5) (CA3 in Van Bulck and Goossens, 2023b) force team i to play at least $\underline{k}(i)$ and at most $\bar{k}(i)$ home games against the teams in subset $\mathcal{T}(i) \subseteq \mathcal{T}$ in each sequence of $\bar{r}(i)$ rounds. Finally, Constraints (5.7) (CA4 in Van Bulck and Goossens, 2023b) impose that the number of home games of teams in $\mathcal{T}'$ against teams in $\mathcal{T}''$ in the rounds of $\mathcal{R}'$ has to be between $\underline{k}$ and $\bar{k}$. These constraints are used, for example, to limit the total number of home games per round between teams that share the same venue. Similar constraints can be imposed in case of away games or games.

- **Game Constraints (GA).**

$$\underline{k} \leq \sum_{(i,j) \in \mathcal{G}'} \sum_{r \in \mathcal{R}'} x_{ijr} \leq \bar{k} \qquad \forall \mathcal{G}' \subseteq \mathcal{G}, \forall \mathcal{R}' \subseteq \mathcal{R}. \tag{5.8}$$

  Given a subset of games $\mathcal{G}' \subseteq \mathcal{G}$ and a subset of rounds $\mathcal{R}' \subseteq \mathcal{R}$, Constraint (5.8) (GA1 in Van Bulck and Goossens, 2023b) imposes a

lower bound $\underline{k}$ and an upper bound $\bar{k}$ on the number of games of $\mathcal{G}'$ that can be played in the rounds of $\mathcal{R}'$.

- **Break Constraints (BR)**. A team $i \in \mathcal{T}$ has a *home/away break* in round $r \in \mathcal{R} \setminus \{0\}$ if i has a home/away game in rounds $r-1$ and r. To model these constraints, we introduce two binary variables y^h_{ir} and y^a_{ir} for each team $i \in \mathcal{T}$ and each round $r \in \mathcal{R} \setminus \{0\}$:

$$y^h_{ir} = \begin{cases} 1 & \text{if team } i \text{ has a home break in round } r \\ 0 & \text{otherwise} \end{cases}$$

and

$$y^a_{ir} = \begin{cases} 1 & \text{if team } i \text{ has an away break in round } r \\ 0 & \text{otherwise.} \end{cases}$$

The break constraints read as follow.

$$y^h_{ir} \geq \sum_{j \in \mathcal{T}(i), j \neq i} x_{ijr} + x_{ijr-1} \qquad \forall i \in \mathcal{T}, \forall r \in \mathcal{R} \setminus \{0\} \tag{5.9}$$

$$y^a_{ir} \geq \sum_{j \in \mathcal{T}(i), j \neq i} x_{jir} + x_{jir-1} \qquad \forall i \in \mathcal{T}, \forall r \in \mathcal{R} \setminus \{0\} \tag{5.10}$$

$$\sum_{r \in \mathcal{R}(i)} y^h_{ir} \leq \bar{k}(i) \qquad \forall i \in \mathcal{T} \tag{5.11}$$

$$\sum_{r \in \mathcal{R}(i)} y^a_{ir} \leq \bar{k}(i) \qquad \forall i \in \mathcal{T} \tag{5.12}$$

$$\sum_{r \in \mathcal{R}(i)} y^h_{ir} + y^a_{ir} \leq \bar{k}(i) \qquad \forall i \in \mathcal{T} \tag{5.13}$$

$$\sum_{i \in \mathcal{T}'} \sum_{r \in \mathcal{R}'} y^h_{ir} + y^a_{ir} \leq \bar{k} \qquad \forall \mathcal{T}' \subseteq \mathcal{T}, \forall \mathcal{R}' \subseteq \mathcal{R}. \tag{5.14}$$

Constraints (5.9) and (5.10) define binary variables y^h_{ir} and y^a_{ir}, respectively. For each team $i \in \mathcal{T}$, Constraint (5.11) (BR1 in Van Bulck and Goossens, 2023b) imposes an upper bound on the number of home breaks of i in the rounds of $\mathcal{R}(i) \subseteq \mathcal{R}$. The same is imposed by Constraints (5.12) for the away breaks and by Constraints (5.13) for the total breaks. Finally, given a subset of teams $\mathcal{T}'$ and a subset of rounds $\mathcal{R}'$, Constraint (5.14) (BR2 in Van Bulck and Goossens, 2023b) fixes the overall number of home and away breaks of teams in $\mathcal{T}'$ in the rounds of $\mathcal{R}'$ to be at most $\bar{k}$.

- **Fairness Constraints (FA)**.

$$-\bar{k}(i,j) \leq \sum_{l \in \mathcal{T} \setminus \{i,j\}} \sum_{r'=1}^{r} x_{ilr'} - x_{jlr'} \leq \bar{k}(i,j) \quad \forall i,j \in \mathcal{T}, i \neq j, \forall r \in \mathcal{R}. \tag{5.15}$$

  For all pair of teams $i,j \in \mathcal{T}$, $i \neq j$ and all rounds $r \in \mathcal{R}$, Constraint (5.15) (FA2 in Van Bulck and Goossens, 2023b) ensures that the difference between the home games played by i and those played by j is at most $\bar{k}(i,j)$ after round r. Analogous constraints can be applied for the away games or games.

- **Separation Constraints (SE)**.

$$\sum_{r' \in \mathcal{R}(i,j)} x_{ijr'} + x_{jir'} \leq 1 - (x_{ijr} + x_{jir}) \quad \forall (i,j) \in \mathcal{G}, i < j, \forall r \in \mathcal{R}, \tag{5.16}$$

  where $\mathcal{R}(i,j) = \{r' \in \mathcal{R} : r - \underline{k}(i,j) \leq r' \leq r + \underline{k}(i,j)\} \cup \{r' \in \mathcal{R} : r' \leq r - \bar{k}(i,j) \vee r' \geq r + \bar{k}(i,j)\}$. Constraints (5.16) (SE1 in Van Bulck and Goossens, 2023b) ensure that if one of the two mutual games (i,j) or (j,i) of two teams $i,j \in \mathcal{T}$ is assigned to round r then the other one cannot be assigned to the rounds of $\mathcal{R}(i,j)$: games (i,j) and (j,i) are separated by at least $\underline{k}(i,j)$ and at most $\bar{k}(i,j)$ rounds.

All constraints presented so far can be handled also in their soft version. Here, we discuss in general terms how their deviation is computed (see Van Bulck et al., 2020, for a detailed description). We remark that all constraints $c \in \mathcal{C}$ have the same structure, i.e. they impose a lower and/or an upper bound on a linear expression:

$$lb_c \leq f_c(x, y^h, y^a) \leq ub_c,$$

where $f_c(x, y^h, y^a)$ is a linear expression in the variables x_{ijr}, y^h_{ir} and y^a_{ir} and lb_c and ub_c are, respectively, a lower and upper bound imposed on $f_c(x, y^h, y^a)$. Except for the fairness and separation constraints, the deviation triggered if one of these constraints is violated is given by the following two constraints

$$d_c \geq lb_c - f_c(x, y^h, y^a) \tag{5.17}$$

$$d_c \geq f_c(x, y^h, y^a) - ub_c. \tag{5.18}$$

For example, if c is a capacity constraint which imposes a lower and an upper bound, respectively $\underline{k}(i)$ and $\bar{k}(i)$, on the number of home games that a team $i \in \mathcal{T}$ can play in the rounds of set $\mathcal{R}(i)$ (Constraint (5.5)), then the deviation triggered by c is equal to the number of home games of i in the rounds of $\mathcal{R}(i)$ less than $\underline{k}(i)$ or more than $\bar{k}(i)$.

Finally, let us discuss how the deviation of the fairness and separation constraints is computed. A fairness constraint limits the difference of home (away or any) games between teams $i \in \mathcal{T}$ and $j \in \mathcal{T}$ after each round $r \in \mathcal{R}$. However, the deviation triggered when it is violated is equal to the maximal difference in home (away or any) games more than $\bar{k}(i,j)$ played by i and j over all the rounds of $\mathcal{R}$ (see Van Bulck and Goossens, 2023b). Hence, to express such deviation, we need to include a non-negative continuous variable z_{ij} storing the maximal difference in home (away or any) games between i and j. For the case of home games, the deviation is computed by including the following constraints.

$$z_{ij} \geq \sum_{l \in \mathcal{T} \setminus \{i,j\}} \sum_{r'=1}^{r} x_{ilr'} - x_{jlr'} \tag{5.19}$$

$$z_{ij} \geq \sum_{l \in \mathcal{T} \setminus \{i,j\}} \sum_{r'=1}^{r} x_{jlr'} - x_{ilr'} \tag{5.20}$$

$$d_c \geq z_{ij} - \bar{k}(i,j). \tag{5.21}$$

Now, let c be a soft version of a separation constraint, which require that the two mutual games of teams $i, j \in \mathcal{T}$, $i < j$ are separated by at least $\underline{k}(i,j)$. The deviation triggered if c is violated has to be equal to the difference between $\underline{k}(i,j)+1$ and the number of rounds between games (i,j) and (j,i):

$$d_c \geq \sum_{r' \in \mathcal{R}(i,j)} |r' - r|(x_{ijr} + x_{jir} + x_{ijr'} + x_{jir'} - 1) \tag{5.22}$$

The case where the two mutual games have to be separated by at most $\bar{k}(i,j)$ rounds can be treated similarly.

5.3 Solution method

We designed a three-stage multi-neighborhood Simulated Annealing[1] for the solution of the problem. The multi-neighborhood is a hexamodal

[1]The source code is available at: `https://github.com/robertomrosati/sa4stt`

neighborhood made up by a portfolio of six different local search neighborhoods, which are specifically tailored for the sports timetabling problems. The metaheuristic method employed for the search of the solution is the slightly modified version of the classical Simulated Annealing defined by Kirkpatrick et al. (1983) described in Chapter 2. The search is executed is three distinct sequential stages, characterized by different parameters values of the metaheuristic and different restriction of the search space. In this section, we explain first of all the general features of the search space and the method employed for the generation of the initial solution. Next, we discuss thoroughly the multi-neighborhood and the Simulated Annealing metaheuristic. Finally, we illustrate the characteristics of the three stages of execution of the algorithm.

5.3.1 Search space

Given the structure of the problem described in Section 5.2, as search space we consider the set of all two-leg round-robin timetables. This means that every possible round-robin timetable, though not necessarily feasible, is a valid solution in the search space. Thus, in every solution, each team plays with every other team twice (home and away), and all teams play exactly one game at every round.

For the instances that require a phased timetable, we allow the algorithm to visit states that break the phase structure. Since the formulation of the problem doesn't provide an explicit phase constraint, we added an artificial tenth cost component, that measures the number of matches that violate the phase requirement. This number is then multiplied for a suitably high weight and the resulting value is assigned to the new cost component. This mechanism, which is applied only to phased instances, make phased violations possible but penalized in the cost function.

A solution is internally described as a matrix of size $|\mathcal{T}| \times |\mathcal{R}|$. Each cell (t, r) contains the index of the opponent of t at the match r. The value is positive if t plays at home at the match r, negative otherwise. Fig. 5.1b provides an example of this encoding, which is used also in the figures in Section 5.3.3 for the explanation of the multi-neighborhood.

5.3.2 Initial solution generation

The initial state can be generated either randomly or through a greedy algorithm. The random procedure that we employ consists in different permutations of teams and rounds on the *canonical pattern* (see, e.g.,

round_0	round_1	round_2	round_3	round_4	round_5	round_6	round_7	round_8	round_9
2 - 0	4 - 0	3 - 0	0 - 5	0 - 1	0 - 3	0 - 4	0 - 2	5 - 0	1 - 0
1 - 3	2 - 1	5 - 1	1 - 4	5 - 2	1 - 5	1 - 2	3 - 1	4 - 1	2 - 5
4 - 5	3 - 5	4 - 2	3 - 2	3 - 4	2 - 4	3 - 5	5 - 4	2 - 3	4 - 3

(a) An example tournament with $|T| = 6$

	round_0	round_1	round_2	round_3	round_4	round_5	round_6	round_7	round_8	round_9
team_0	-2	-4	-3	+5	+1	+3	+4	+2	-5	-1
team_1	+3	-2	-5	+4	-0	+5	+2	-3	-4	+0
team_2	+0	+1	-4	-3	-5	+4	-1	-0	+3	+5
team_3	-1	-5	+0	+2	+4	-0	+5	+1	-2	-4
team_4	+5	+0	+2	-1	-3	-2	-0	-5	+1	+3
team_5	-4	+3	+1	-0	+2	-1	-3	+4	+0	-2

(b) Internal representation of the previous tournament

Figure 5.1: Example of the internal solution representation of a timetable for a round-robin tournament with 6 teams and 10 games: the upper part is the listing of the actual games in the tournament while the lower part reports its encoding.

de Werra, 1981). It produces a double round-robin tournament, but it does not provide any feasibility guarantee regarding the hard constraints of the problem: Those are then restored by the Simulated Annealing procedure.

Given an input instance with $|\mathcal{T}|$ teams and $|\mathcal{R}|$ rounds, the random initial solution is generated performing the following steps:

1. A single-leg canonical pattern for the $|\mathcal{T}|$ teams with $|\mathcal{R}|/2$ rounds is generated. Each team meets every other team exactly once.

2. A random permutation is performed on the $|\mathcal{T}|$ teams.

3. The timetable is mirrored in order to obtain a two-leg tournament. At this moment, the second leg is identical to the first one, except for the home-away order that is inverted.

4. If the instance is not phased, a random permutation is executed on the $|\mathcal{R}|$ rounds. Otherwise, if the instance is phased, two random permutations are executed. The first one involves the rounds $\{0, \ldots, |\mathcal{R}|/2 - 1\}$, the second one is performed on the rounds $\{|\mathcal{R}|/2, \ldots, |\mathcal{R}|-1\}$. Hence, the initial random solution does not violate the phase constraint.

Also the greedy algorithm is based on the canonical pattern, which is used as a reference for the constructive steps. The idea behind the greedy algorithm is to generate and test the addition of candidate rounds that are constructed on the basis of a reference tournament of *template rounds* obtained as in the random procedure. These rounds are templates instead of actual ones since all their possible perturbations, according to some of the symmetries that are inherent in round-robin tournaments, are produced in the generation process. In detail, the symmetries used are those among rounds (i.e., permuting the order of the rounds does not violate the round-robin tournament property), and those among the venues of each game.

Starting from an empty initial solution, the greedy process selects, at each step, the best candidate to be added to the current solution according to its contribution to constraint violations. Since the solution is incomplete, the check is restricted to only those constraints that can be (at least partially) evaluated in the current partial solution once it has been extended with any of the candidate rounds.

In Fig. 5.2 we exemplify a step of the greedy process in case of $|T| = 6$ teams in a non-phased setting. In the example, the current solution consists of four assigned rounds and six remaining template rounds (denoted by χ_i), which are still available from the initial canonical pattern. Different situations arise, for instance, in the generation of round candidates for the χ_1 and χ_2 templates.

As for the template χ_1, which has not been included yet in the solution, there is full freedom in deciding the home-away status of the games, therefore all possible venues permutations can be generated and evaluated. Conversely, since one copy of the χ_2 template has been already included in the solution (namely in round 1), among all the possible venues permutations only the one that mirrors the already included copy of the template is possible. This is however inherent in the fact that the reference tournament for the round templates is created through a concatenation of two single round-robin canonical patterns.

Each generated round candidate is tried for the completion of the current solution and the partial cost of this addition is computed. For example, in the figure, the constraint BR1[2] reported above the current solution requires that no more than 2 *breaks* occur for team 5 during periods 1–5. This constraint can be partially checked for its cost (which is zero, since there are no more

[2]Refer to (Van Bulck and Goossens, 2023b) for the comprehensive explanation of all constraints employed in the competition.

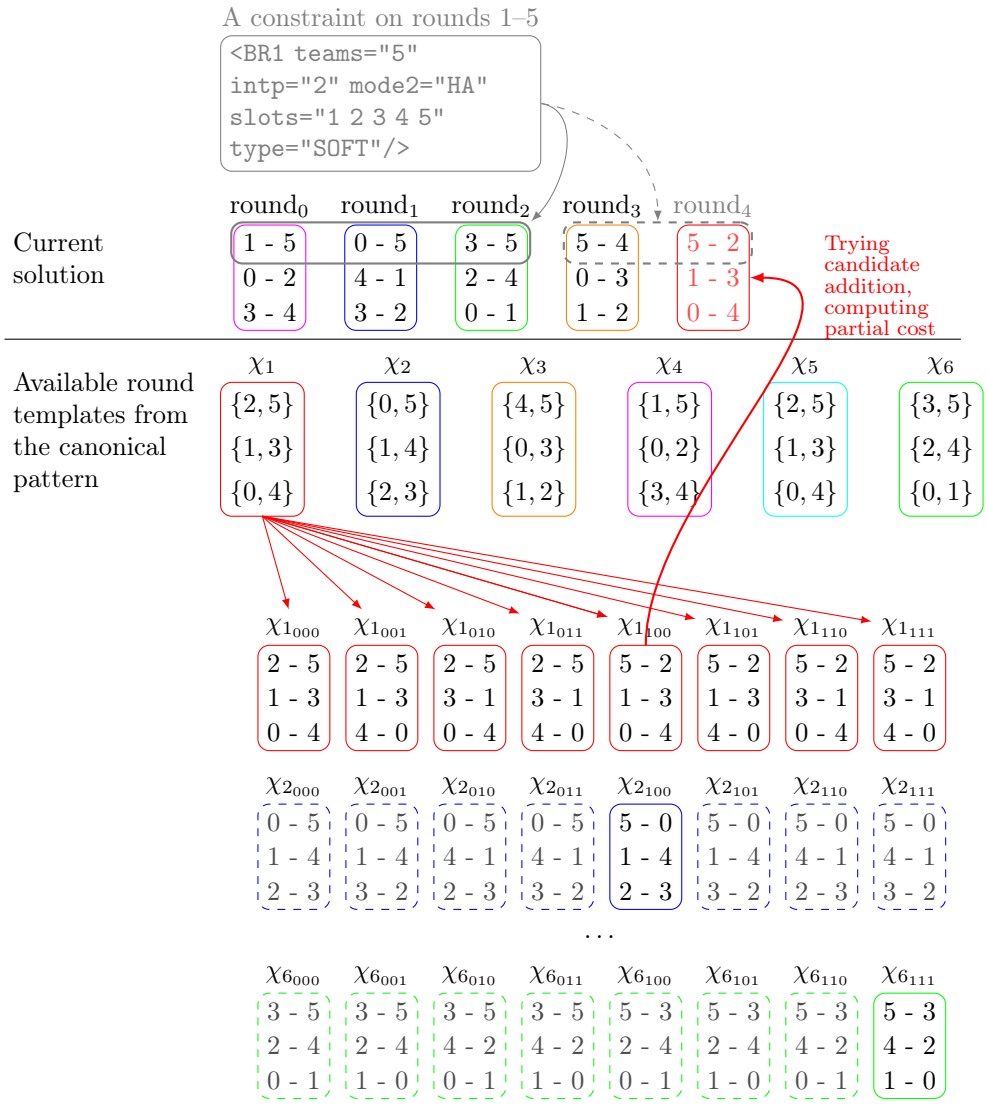


Figure 5.2: One step of the greedy constructive procedure in the case of a non phased tournament: the procedure tries to complete the partial solution with the best possible combination of template assignment and game venues.

than 2 breaks) for the periods 1–3 already added to the solution, whereas it cannot be checked yet for period 4 and the possible candidate addition.

Among all the possible candidates, the one that achieves the minimum

(partial) cost is selected and the corresponding template is removed from the set of available ones. Possible ties in the cost value are randomly broken.

In the case of a phased tournament the greedy procedure is adapted to ensure that the two legs of the tournament are separated. In order to achieve this goal, the tournament used as a reference for the first leg consists of a single round-robin tournament template and after the first leg is completed the second leg is constructed with another (possibly different, in terms of team permutations) single round-robin tournament using the full generation of combinations but pruning those that overlap with the games already included in the first leg.

Finally, to obtain numerous diverse initial solutions also with the greedy procedure, the indexes of the teams are randomly permuted. That is, before starting the process, a random permutation of the indexes is drawn and the mapping between the teams in the candidate round (i.e., the logical indexes) and the actual teams is computed by applying this permutation. To enhance the randomness, in the case of the phased tournament two distinct permutations are computed for the first leg and the second leg assignments.

As discussed further in Section 5.4.2, this choice seems to mildly outperform the random initial solution. Nevertheless, the improvement margin is not considerably large, so we decided to keep both possibilities in our algorithm, leaving to the user the choice of the start method through an input parameter.

5.3.3 Multi-neighborhood

The core component of the solution method is our multi-neighborhood composed by the union of six different neighborhoods. Five of them, called SwapHomes, SwapTeams, SwapRounds, PartialSwapTeams, and PartialSwapRounds, are adaptations of classical ones from Anagnostopoulos et al. (2006), Ribeiro and Urrutia (2007), and Di Gaspero and Schaerf (2007). In addition, we introduce a novel neighborhood called PartialSwapTeamsPhased, specifically designed to deal with phased instances. Experimental results highlight that the usage of the novel neighborhood allows us to reach better solutions in terms of objective function and to achieve feasibility on certain large phased instances, which would be, otherwise, very hard to tackle. All the neighborhoods ensure that the double round-robin structure of the tournament is conserved, but they don't provide any guarantees on the feasibility of the solution.

The multi-neighborhood is designed to be employed by the Simulated Annealing metaheuristic, described in Section 5.3.4, that randomly draws a

move from the multi-neighborhood at every iteration. A desirable feature of the multi-neighborhood is to give higher frequency of execution to those moves that belong to neighborhoods that, on average, lead to the most significant improvements of the solution. So, an essential property of the multi-neighborhood is that each neighborhood is associated with a probability σ. The draw of the move in the Simulated Annealing is then executed into two steps. The first step is the random selection of one of the six neighborhoods, according to the given probabilities. The second step is the random selection of a move inside the neighborhood. The probability of the move inside the neighborhoods is shaped as a uniform random variable.

The values of the probabilities are defined through a tuning procedure discussed in Section 5.4.2, whilst the six neighborhoods and their specifications are illustrated hereafter. We employ graphics to display examples of moves, in which the matrix on the left shows the state before the move, the one on the right represents the new state. Changes are marked in bold and colored.

SwapHomes

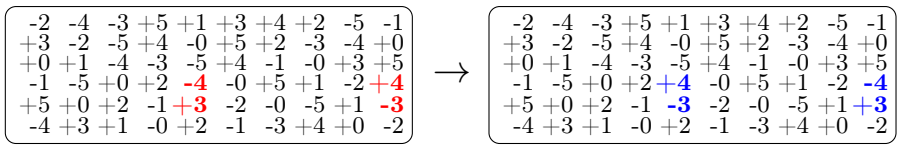


Figure 5.3: Execution of the SwapHomes move $\mathsf{SH}\langle 3, 4\rangle$.

The move SwapHomes takes as attributes two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and it is denoted as $\mathsf{SH}\langle t_i, t_j\rangle$. It swaps the home/away position of the two games between t_i and t_j. Fig. 5.3 shows the execution of the move.

SwapTeams

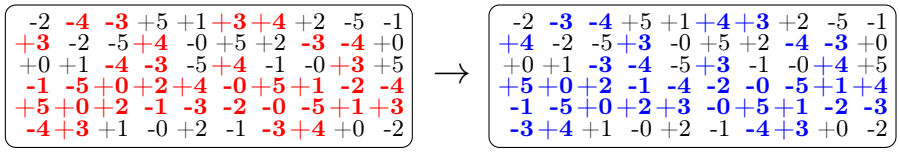


Figure 5.4: Execution of the SwapTeams move $\mathsf{ST}\langle 3, 4\rangle$.

The move SwapTeams takes as attributes two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and it is denoted as $\mathsf{ST}\langle t_i, t_j \rangle$. It swaps the positions of t_i and t_j throughout the whole timetable. Fig. 5.4 shows the execution of the move.

SwapRounds

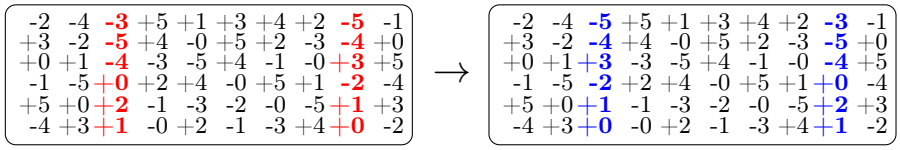


Figure 5.5: Execution of the SwapRounds move $\mathsf{SR}\langle 2, 8 \rangle$.

The move SwapRounds takes as attributes two rounds $r_i, r_j \in \mathcal{R}$, $r_i \neq r_j$, and it is denoted as $\mathsf{SR}\langle r_i, r_j \rangle$. It swaps the two rounds in the timetable. That is to say, all the matches assigned to r_i are moved to r_j, and vice versa. Fig. 5.5 shows the execution of the move.

PartialSwapTeams

The move PartialSwapTeams takes as attributes two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and a set of rounds $\mathcal{R}_s = \{r_1, \ldots, r_s\}$, $\mathcal{R}_s \subset \mathcal{R}$. The move is denoted as $\mathsf{PST}\langle t_i, t_j, \mathcal{R}_s \rangle$. It swaps the positions of t_i and t_j on the set of rounds in $\mathcal{R}_s$. As the name suggests, it works similarly to SwapTeams, with the main difference that the move is not executed on the whole timetable, but only on a subset of rounds.

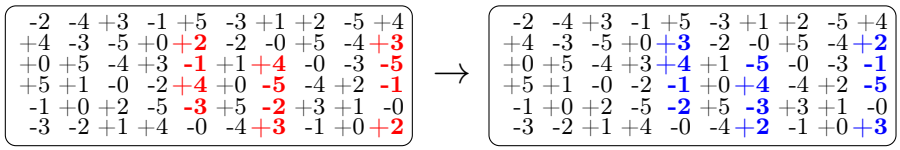


Figure 5.6: Execution of the PartialSwapTeams move $\mathsf{PST}\langle 2, 3, \{4, 6, 9\} \rangle$.

A fundamental requirement for the construction of $\mathcal{R}_s$ is that each team t_k, playing against teams t_i, t_j in the rounds in $\mathcal{R}_s$, must play against both t_i and t_j exactly the same amount of times. If $\mathcal{R}_s$ satisfies the precondition, the swap of t_i and t_j in the rounds in $\mathcal{R}_s$ leads to another correct round-robin timetable. In practice though, large subsets $\mathcal{R}_s$ are not particularly desirable, because they considerably slow down the generation of the move with a negative impact on the overall time performance of the algorithm. For this

reason, we impose a limitation on the maximal size of the set $\mathcal{R}_s$ during the move generation procedure. Fig. 5.6 shows the execution of the move.

PartialSwapRounds

The move PartialSwapRounds takes as attributes two rounds $r_i, r_j \in \mathcal{R}$, $r_i \neq r_j$, and a set of teams $\mathcal{T}_s = \{t_1, \ldots, t_s\}$, $\mathcal{T}_s \subset \mathcal{T}$. The move is denoted as $\mathsf{PSR}\langle \mathcal{T}_s, r_i, r_j \rangle$. It produces the swap between r_i and r_j of the matches including teams in $\mathcal{T}_s$. As the name suggests, it works similarly to PartialRounds, with the main difference that the move is not executed on the whole set of matches in the two rounds, but only on a subset of matches.

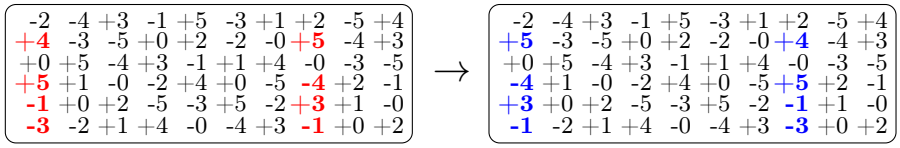


Figure 5.7: Execution of the PartialSwapRounds move $\mathsf{PSR}\langle\{1, 5, 3, 4\}, 0, 7\rangle$.

A fundamental requirement for the construction of $\mathcal{T}_s$ is that every team $t_k \in \mathcal{T}_s$ plays only with teams from $\mathcal{T}_s$ in the two rounds r_i and r_j. In practice though, large subsets $\mathcal{T}_s$ are not particularly desirable, because they considerably slow down the generation of the move with a negative impact on the overall time performance of the algorithm. For this reason, we impose a limitation on the maximal size of the set $\mathcal{T}_s$ during the generation of the move. Fig. 5.7 shows the execution of the move.

PartialSwapTeamsPhased

The move PartialSwapTeamsPhased is a novel neighborhood that we designed with the main motivation to deal with the phased version of the problem. The five neighborhoods discussed so far, indeed, work well on the non-phased instances, but turned out to be insufficient for obtaining good results on the phased ones. The neighborhood PartialSwapTeams, in particular, has quite disruptive side effects on the phase structure of the timetable, that are only sporadically beneficial to the search. PartialSwapTeamsPhased, on the other hand, allows to reach new solutions through partial swaps of teams without variations of the current state of the phase. We discuss in this section the fundamentals of the neighborhood, and we forward the reader to Section 5.4.4 for an analysis of the experimental data.

As the name suggests, PartialSwapTeamsPhased takes inspiration from the above-mentioned PartialSwapTeams. An example of the disruptive effect of PartialSwapTeams on the phase structure is given in Fig. 5.6. Before the execution of the move, the phase structure is respected. After the move is applied, both matches between teams 2 and 4 take place during the first phase of the tournament, and both matches between teams 3 and 4 take place in the second phase of the tournament, which constitute two violations of the phase constraint. To overcome this issue, the new neighborhood PartialSwapTeamsPhased makes use of a new concept of mixed phase, that allows the new move to be invariant with respect to the phase.

We define as *mixed phase* of a two-leg round-robin tournament the partition of the timetable in two subsets, named mixed legs, where each couple of teams play together, respectively, for the first and for the second time. Hence, the first mixed leg is the set of all the matches where teams meet with each other for the first time, the second mixed leg is the set of all the matches where teams meet for the second time. This definition is independent from the current satisfaction of the phase constraint. It might happen that mixed phase and actual phase correspond: this is the case when the phased constraint is respected.

-2	-4	-3	+5	+1	+3	+4	+2	-5	-1
+3	-2	-5	+4	-0	+5	+2	-3	-4	+0
+0	+1	-4	-3	-5	+4	-1	-0	+3	+5
-1	-5	+0	+2	+4	-0	+5	+1	-2	-4
+5	+0	+2	-1	-3	-2	-0	-5	+1	+3
-4	+3	+1	-0	+2	-1	-3	+4	+0	-2

$\leftrightarrow$

0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1

(a) Mixed phase and actual phase correspond.

-4	-5	+3	+4	-3	+2	+1	-1	-2	+5
-2	-3	+2	+5	-5	+3	-0	+0	-4	+4
+1	-4	-1	-3	+4	-0	-5	+5	+0	+3
-5	+1	-0	+2	+0	-1	+4	-4	+5	-2
+0	+2	+5	-0	-2	-5	-3	+3	+1	-1
+3	+0	-4	-1	+1	+4	+2	-2	-3	-0

$\leftrightarrow$

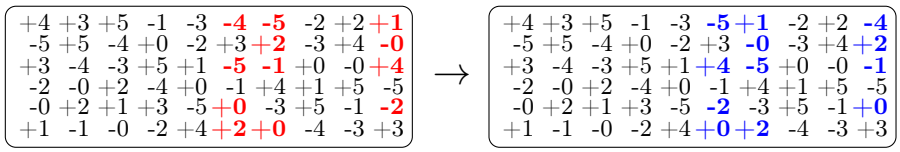
0	0	0	1	1	0	0	1	1	1
0	0	1	0	1	1	0	1	0	1
0	0	1	0	1	0	0	1	1	1
0	0	0	0	1	1	0	1	1	1
0	0	0	1	1	1	0	1	0	1
0	0	0	0	1	1	0	1	1	1

(b) Mixed phase and actual phase differ, the phase constraint is not respected.

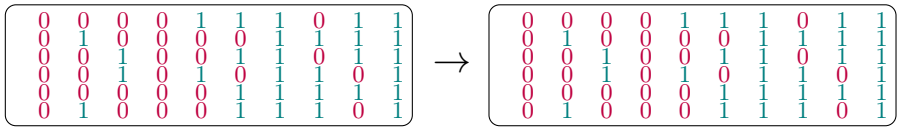
Figure 5.8: Mixed phase and mixed legs concepts.

Fig. 5.8 visually explains the concept. Both figures contain two boxes: the box on the left is the representation of the current tournament timetable, the box on the right is the corresponding mixed phase. The first and second mixed legs are denoted, respectively, by means of zeros and ones. Fig. 5.8a describes the situation of a timetable that satisfies the phase constraint, and Fig. 5.8b represents a timetable where the phase constraint is violated. Matches where teams meet each other for the first time belong to the first

mixed leg and are denoted by zeros in the box on the right, matches where teams meet each other for the second time belong to the second mixed leg and are denoted by ones.



(a) Move execution



(b) The mixed phase is not modified.

Figure 5.9: Execution of the PartialSwapTeamsPhased move PSTP $\langle 0, 2, \{5, 6, 9\}\rangle$.

The move PartialSwapTeamsPhased takes as attribute two teams $t_i, t_j \in \mathcal{T}$, $t_i \neq t_j$, and a set of rounds $\mathcal{R}_s = \{r_1, \ldots, r_s\}$, $\mathcal{R}_s \subset \mathcal{R}$. The essential prerequisite is that the matches involved in the move must all belong to the same mixed leg, according to the definition provided. The move is denoted as PSTP$\langle t_i, t_j, \mathcal{R}_s\rangle$, and, similarly to the move PartialSwapTeams, it produces the swap of the positions of t_i and t_j on the set of rounds in $\mathcal{R}_s$. Consequently, the maximum size of the set $\mathcal{R}_s$ corresponds to the size of a mixed leg, which is $|\mathcal{R}_s| \leq |\mathcal{R}|/2$. Differently from PartialSwapTeams, t_i and t_j keep their home/away positions against same opponents. An example of move execution is given in Fig. 5.9. To make more evident the relation with the mixed phase, we show it for a solution that violates the phased constraint.

5.3.4 Metaheuristic

The metaheuristic employed is the Multi-Neighborhood Simulated Annealing described in Chapter 2, that we run in three stages.

The algorithm is structured into three distinct stages, that consist in three independent runs of the Simulated Annealing procedure. The first stage starts its search either from a random or from a greedy solution, the second and the third stages are warm-started with the output of the previous stage as initial solution. The differences between the stages consist in the restrictions applied to the search space, and in the exclusion or inclusion of certain constraints (see Table 5.1).

Table 5.1: Description of the features of the three stages.

	Constraints	
	Hard	Soft
Stage 1	As cost component	Not used
Stage 2	As cost component	As cost component
Stage 3	Violation not allowed	As cost component

Stage 1: Hard constraints are included in the cost function, soft constraints are not considered. The goal of Stage 1 is to rapidly find a feasible solution. In our experiments, this stage shows its effectiveness specifically on large phased instances, where the metaheuristic faces most problems in finding a feasible timetable.

Stage 2: All constraints are used and both feasible and infeasible regions are explored. The costs associated to hard constraints and phased cost component are multiplied by suitable weights. The goal of Stage 2 is to find a good solution in terms of objective function.

Stage 3: All constraints are used, but moves that violate hard constraints are not allowed and only the feasible region is explored. The goal of Stage 3 is to reach the best possible solution that can be achieved from the best state found in Stage 2, through the exploration of the feasible region only. If the outcome of the previous stages is not a feasible solution, Stage 3 is not performed.

5.4 Experimental results

In this section, we discuss the experimental setting of the Simulated Annealing algorithm and the results obtained by applying it to the instances proposed for the ITC2021 competition. We also assess the performances of Model $\mathcal{M}$ by running it on CPLEX within a time limit. For these latter experiments, we consider the instances of the competition and some others of smaller size obtained by performing reductions on the competition instances.

Our code was developed in C++ and compiled with GNU g++ version 9.3.0 in `-O3` mode, on Ubuntu 20.04.2 LTS. All the experiments presented in this section were run on a machine equipped with AMD Ryzen Threadripper PRO 3975WX processor with 32 cores, hyper-threaded to 64 virtual cores,

Table 5.2: List of instances from the ITC2021 competition with indication of some general features and overall number of hard and soft constraints.

Instance	Features				Instance	Features			
	Phased	Teams	Hard	Soft		Phased	Teams	Hard	Soft
Early_1	yes	16	83	113	Middle_9	no	18	94	201
Early_2	yes	16	53	114	Middle_10	yes	20	198	714
Early_3	yes	16	148	186	Middle_11	yes	20	176	1048
Early_4	yes	18	164	268	Middle_12	yes	20	63	241
Early_5	yes	18	207	587	Middle_13	no	20	219	350
Early_6	yes	18	192	797	Middle_14	no	20	63	817
Early_7	no	18	175	1159	Middle_15	no	20	95	133
Early_8	no	18	70	582	Late_1	no	16	235	542
Early_9	no	18	90	102	Late_2	no	16	246	1077
Early_10	yes	20	246	1015	Late_3	no	16	127	439
Early_11	no	20	246	1108	Late_4	yes	18	96	34
Early_12	yes	20	179	35	Late_5	yes	18	176	747
Early_13	no	20	100	432	Late_6	yes	18	163	159
Early_14	no	20	56	56	Late_7	no	18	126	738
Early_15	no	20	187	1224	Late_8	yes	18	110	195
Middle_1	yes	16	144	993	Late_9	no	18	102	402
Middle_2	yes	16	246	1231	Late_10	yes	20	233	694
Middle_3	no	16	237	1212	Late_11	yes	20	52	366
Middle_4	yes	18	97	168	Late_12	no	20	244	1009
Middle_5	yes	18	151	197	Late_13	no	20	169	134
Middle_6	yes	18	162	154	Late_14	no	20	116	993
Middle_7	no	18	141	476	Late_15	no	20	51	41
Middle_8	no	18	62	224					

with base clock frequency of 3.5 GHz, and 64 GB of RAM. In both settings, one single virtual core is used for each experiment.

5.4.1 Instances

The algorithm was run on the 45 instances of the competition, that are available for download on the website of the competition (Van Bulck and Goossens, 2023b). We perform a general analysis of their main features. The size, expressed in number of teams, is always comprised between 16 and 20 teams, and 22 instances in total have a phased requirement. As we can observe in Table 5.2, the total number of hard and soft constraints fluctuates considerably among instances. The range for hard constraints goes from a minimum of 51 in instance Late_11 to a maximum of 246 in instances Early_10, Early_11, Middle_2, and Late_2. The range for soft constraints goes from a minimum of 34 in instance Late_4 to a maximum of 1231 in instance Middle_2. Incidentally, instance Middle_2, which has the highest overall number of constraints, 1477, is also the only instance in which

Table 5.3: Details of the number of constraints in each instance.

Name	CA1		CA2		CA3		CA4		BR1		BR2		GA1		SE1		FA2	
	H	S	H	S	H	S	H	S	H	S	H	S	H	S	H	S	H	S
E1	25	17	10	0	0	0	0	84	35	0	1	0	12	10	0	1	0	1
E2	38	30	0	0	2	82	0	0	12	0	1	0	0	1	0	0	0	1
E3	24	0	72	21	0	112	0	0	18	0	0	1	34	51	0	0	0	1
E4	0	32	0	235	0	0	85	0	44	0	1	0	34	0	0	1	0	0
E5	41	27	36	331	2	111	81	117	23	0	1	0	23	0	0	1	0	0
E6	38	31	71	591	2	54	81	115	0	0	0	1	0	3	0	1	0	1
E7	42	31	30	620	1	112	84	340	12	0	1	0	5	55	0	1	0	0
E8	19	0	8	57	0	112	0	339	39	0	0	0	4	73	0	0	0	1
E9	39	0	14	0	0	88	0	0	23	10	0	1	14	2	0	0	0	1
E10	42	32	72	620	2	23	85	339	44	0	1	0	0	0	0	1	0	0
E11	42	32	72	620	2	112	85	340	44	0	1	0	0	3	0	1	0	0
E12	37	0	72	0	2	20	31	0	20	13	0	1	17	1	0	0	0	0
E13	41	31	27	257	1	110	0	0	20	10	1	0	10	24	0	0	0	0
E14	5	30	0	0	0	0	0	0	17	24	0	1	34	0	0	0	0	1
E15	42	0	72	620	2	112	71	340	0	24	0	1	0	126	0	0	0	1
M1	0	32	14	620	0	0	85	340	44	0	1	0	0	0	0	1	0	0
M2	42	32	72	620	2	112	85	340	44	0	1	0	0	126	0	1	0	0
M3	42	0	72	617	2	107	85	338	36	21	0	1	0	126	0	1	0	1
M4	31	18	17	0	1	0	0	41	23	24	0	0	25	85	0	0	0	0
M5	41	24	40	33	0	12	0	0	44	0	0	1	26	126	0	0	0	1
M6	39	0	41	0	2	111	30	27	44	13	0	1	6	1	0	1	0	0
M7	0	30	0	355	1	0	78	51	28	21	0	1	34	17	0	1	0	0
M8	16	0	0	27	2	108	0	34	32	14	0	0	12	41	0	0	0	0
M9	42	0	0	37	1	100	19	39	28	23	0	1	4	0	0	0	0	1
M10	41	15	71	363	0	0	46	262	39	0	1	0	0	74	0	0	0	0
M11	7	0	71	612	2	88	84	340	12	0	0	0	0	7	0	0	0	1
M12	0	32	28	168	1	13	0	0	29	21	0	1	5	4	0	1	0	1
M13	42	29	72	242	1	0	85	76	12	0	0	0	7	2	0	1	0	0
M14	5	18	11	319	2	112	0	338	38	10	1	0	6	19	0	0	0	1
M15	12	0	23	0	0	77	0	0	37	10	0	1	23	44	0	1	0	0
L1	42	32	72	198	1	13	82	283	38	0	0	0	0	15	0	0	0	1
L2	42	0	72	620	2	112	85	340	44	0	1	0	0	5	0	0	0	0
L3	42	0	72	326	1	43	0	60	12	0	0	1	0	7	0	1	0	1
L4	0	32	0	0	0	0	18	0	44	0	0	0	34	1	0	1	0	0
L5	0	19	69	614	2	0	81	109	0	0	1	0	23	4	0	0	0	1
L6	0	32	0	125	0	0	85	0	44	0	0	1	34	0	0	1	0	0
L7	42	32	40	601	1	61	0	0	37	0	1	0	5	43	0	1	0	0
L8	37	15	0	14	0	111	0	0	41	24	0	1	32	29	0	1	0	0
L9	40	0	20	250	2	112	0	0	40	20	0	1	0	18	0	0	0	1
L10	0	31	67	447	2	0	85	205	44	0	1	0	34	10	0	1	0	0
L11	6	0	16	274	0	88	0	0	12	0	1	0	17	3	0	0	0	1
L12	40	32	72	620	2	16	85	340	44	0	1	0	0	0	0	1	0	0
L13	14	32	72	15	2	0	81	71	0	0	0	1	0	13	0	1	0	1
L14	42	0	72	390	2	112	0	340	0	24	0	0	0	126	0	0	0	1
L15	5	0	0	0	0	15	0	0	12	24	0	1	34	0	0	0	0	1

our solver was not able to determine any feasible solution. More in general, from our experimental results we observed that most instances where our solver shows the most deficiencies are also among those characterized by many hard constraints, but only when also the phase requirement is present. For this reason, the solver redistributes the total number of iterations in favor of Stage 1 when it recognizes a phased instance with a quantity of hard constraints above a certain threshold. Table 5.3 provides additional details on the cardinality of constraints of each type in their hard and soft versions. It is possible to notice that constraint types BR2, FA2 and SE1 are always expressed uniquely, if present, so only six constraint types out of nine are actually declined into multiple constraints. Finally, it is noteworthy to mention that each individual constraint involves different quantities of teams or slots, so that also the individual contribution to the instance complexity may differ substantially.

5.4.2 Parameters and tuning

For the three-stage multi-neighborhood Simulated Annealing to be effective on the given instances, several parameters need to be tuned. In this work, we tuned the probabilities σ_{SH}, σ_{ST}, σ_{SR}, σ_{PST}, σ_{PSR}, σ_{PSTP} of the six neighborhoods separately on the phased and on the non-phased instances. These probabilities are stage-independent. The specific parameters of the Simulated Annealing metaheuristics, on the other hand, were tuned separately for each stage. For the Simulated Annealing we decided to tune only the start temperature and the final temperature, which turned out to be the most critical parameters from previous research work (see, e.g., Bellio et al., 2016, 2021). Conversely, we fixed the sample acceptance ratio and the cooling rate values to consolidated and robust values found in previous work. In our algorithm, we assigned weights to the different hard cost components and these weights also underwent a tuning procedure. They were employed in Stage 1 and Stage 2, since moves that violate feasibility are not considered in Stage 3. Finally, the decision whether to use a random or a greedy start for Stage 1 was also subject to a tuning procedure.

As introduced in Section 5.3.4, we allow in Stage 1 and Stage 2 the exploration of the infeasible region and, for phased instances, also the break of the phase structure. Thus, the weights assigned to hard violations and phased violations also require tuning. In this case, we didn't only search for possible values, but we compared two different approaches: feature-based or fixed values. Specifically, the only feature that we take into account for this problem is the number of hard constraints in the given instance. Let N_h

be the number of hard constraints, w_h and w_p, respectively, the hard cost component weight and the phased cost component weight, and k a generic constant. Eqs. (5.23) and (5.24) describe how we can obtain the weights for the hard cost component and the phased cost component from the number of hard constraints, in the feature-based scenario. The value of k is also determined through a tuning procedure. Please note that k is float-valued and w_h and w_p are integer-valued, so a rounding is applied. In our tuning procedure, the feature-dependent approach resulted to be the most effective for Stage 2, while for Stage 1 fixed values resulted to be more suitable.

$$w_h = N_h \cdot k \tag{5.23}$$

$$w_p = N_h \cdot k \cdot w_h \tag{5.24}$$

The whole tuning procedure was performed with the aid of the tool JSON2RUN, described in Chapter 4.

Table 5.4 contains the list of the parameters and related values. The column *Tuning range* contains the lower and upper bounds of the ranges taken into account by the tuning procedure, which don't constitute a boundary in our algorithm. The dual comparisons aimed to determine whether to use a random or a greedy start and whether to use fixed or feature-dependent w_h and w_p don't require a Hammersley sampling. The outcome of the tuning procedure is shown under the columns *Assigned values*.

Finally, Table 5.5 presents the outcome of a dual comparison between the random start and the greedy start. We limited the execution of the algorithm to one million iterations, which we consider a short run, of Stage 1, exclusively. The results are given in terms of feasibility ratio, because in this stage of the algorithm we are not focused yet in optimizing the objective function. In general, we can observe that the greedy start ensures a higher probability of finding a feasible solution, at a price of a slightly longer execution time.

5.4.3 Analysis of the results

We report in this section an overview of the experimental results.

First, we discuss the results obtained on Model $\mathcal{M}$. We used the solver CPLEX 20.1 and we imposed different time limits, ranging from one hour to 24 hours. We employed one single CPU per run.

Solving the model on the competition instances did not yield good results: within a time limit of one hour, a feasible solution was found only for instance

Table 5.4: Parameter tuning

Parameter	Description	Tuning Range	Assigned Values		
			Not Phased		Phased
σ_{SH}	SwapHomes	$[0.0, 1.0]$	0.154		0.130
σ_{ST}	SwapTeams	$[0.0, 1.0]$	0.070		0.020
σ_{SR}	SwapRounds	$[0.0, 1.0]$	0.025		0.080
σ_{PST}	PartialSwapTeams	$[0.0, 1.0]$	0.319		0.120
σ_{PSR}	PartialSwapRounds	$[0.0, 1.0]$	0.350		0.520
σ_{PSTP}	PartialSwapTeamsPhased	$[0.0, 1.0]$	0.070		0.130
			Stage 1	Stage 2	Stage 3
T_0	Start Temperature	$[0, 2000]$	179	600	17.9
T_{min}	End Temperature	$[0, 20]$	2.1	3.52	0.21
w_{h,ca_1}	Weight of CA1 hard	$[1, 10]$	7	7	-
w_{h,ca_2}	Weight of CA2 hard	$[1, 10]$	8	8	-
w_{h,ca_3}	Weight of CA3 hard	$[1, 10]$	2	2	-
w_{h,ca_4}	Weight of CA4 hard	$[1, 10]$	8	8	-
w_{h,ga_1}	Weight of GA1 hard	$[1, 10]$	10	10	-
w_{h,br_1}	Weight of BR1 hard	$[1, 10]$	1	1	-
w_{h,br_2}	Weight of BR2 hard	$[1, 10]$	6	6	-
w_{h,fa_2}	Weight of FA2 hard	$[1, 10]$	1	1	-
w_{h,se_1}	Weight of SE1 hard	$[1, 10]$	1	1	-
	Initial solution	$\{random, greedy\}$	*greedy*	-	-
w_h	Feature-dependent w_h	$\{yes, no\}$	*no*	*yes*	-
w_p	Feature-dependent w_p	$\{yes, no\}$	*no*	*yes*	-
k	Correlation factor	$[0.1, 1000.0]$	-	0.5	-
w_h	Hard weight (fixed)	$[1, 1000]$	10	-	-
w_p	Phased weight (ixed)	$[1, 1000]$	117	-	-

Table 5.5: Comparison of the results obtained by the random start and the greedy start on 30 short runs of Stage 1.

Instance	Random		Greedy		Instance	Random		Greedy	
	Feas.	Time (s)	Feas.	Time (s)		Feas.	Time (s)	Feas.	Time(s)
Early_1	1.00	4.44	1.00	5.03	Middle_9	1.00	10.49	1.00	12.96
Early_2	0.87	26.50	0.97	24.40	Middle_10	0.00	20.41	0.00	25.90
Early_3	1.00	2.24	1.00	2.72	Middle_11	0.83	64.05	0.90	69.47
Early_4	0.00	22.90	0.00	25.03	Middle_12	1.00	14.39	1.00	18.98
Early_5	0.00	73.41	0.00	76.77	Middle_13	1.00	25.04	1.00	36.09
Early_6	0.20	68.10	0.20	70.42	Middle_14	1.00	26.85	1.00	34.05
Early_7	0.33	40.14	0.30	44.24	Middle_15	1.00	1.67	1.00	6.91
Early_8	1.00	1.23	1.00	2.87	Late_1	0.97	21.75	1.00	20.89
Early_9	1.00	1.29	1.00	3.08	Late_2	0.00	58.60	0.00	60.31
Early_10	0.00	92.14	0.00	102.42	Late_3	1.00	7.49	1.00	8.07
Early_11	0.37	83.88	0.60	90.42	Late_4	1.00	3.11	1.00	4.50
Early_12	0.47	73.86	0.43	78.61	Late_5	0.00	73.04	0.00	76.02
Early_13	1.00	16.09	1.00	22.23	Late_6	1.00	7.15	1.00	9.64
Early_14	1.00	0.94	1.00	5.73	Late_7	1.00	13.66	1.00	16.13
Early_15	1.00	46.00	1.00	56.23	Late_8	1.00	2.78	1.00	3.91
Middle_1	0.00	23.55	0.00	24.47	Late_9	1.00	24.39	1.00	26.07
Middle_2	0.00	60.07	0.00	61.46	Late_10	0.00	90.38	0.00	99.97
Middle_3	0.00	58.68	0.00	60.33	Late_11	1.00	4.16	1.00	7.99
Middle_4	1.00	12.18	1.00	14.45	Late_12	0.87	66.50	0.87	80.68
Middle_5	1.00	3.26	1.00	4.55	Late_13	1.00	46.21	1.00	56.76
Middle_6	0.27	59.18	0.40	59.57	Late_14	1.00	31.55	1.00	39.43
Middle_7	1.00	19.80	1.00	23.17	Late_15	1.00	0.80	1.00	5.71
Middle_8	1.00	20.99	1.00	23.68					

Late_4, the other 44 instances were left unsolved. Longer time limits provided very limited improvements. Within 24 hours, feasible solutions were found only for six instances (E8, M4, M8, M15, L4, L8), with values of the objective function far from those obtained by the Simulated Annealing within analogous running times. Hence, to assess the performances and the limits of Model $\mathcal{M}$, we run tests on two clusters of instances obtained by reducing the size of the competition ones. In the first cluster, we removed constraint types and pairs of constraint types from each competition instance which contains them. The columns of Table 5.6 (left) report respectively: the removed constraint types; the number of reduced instances; the number of instances for which a feasible, but not optimal, solution is found; the number of instances solved to optimality. From Table 5.6 (left), we infer that the solver still struggles to produce feasible solutions when only one constraint type is removed. Removing pairs of constraints seems to bring benefits only when the capacity and break constraints are both removed: the solver manages to provide 26 feasible solutions for the considered instances, six of which are proven to be optimal in an average time of 30 seconds. This is in line with the structure of the instances, indeed, several of them consider many capacity and break constraints in their hard version (see Table 5.3).

In the second cluster of instances, we reduced the size of the competition instances in terms of number of teams. Specifically, we restrict the cardinality of team set $\mathcal{T}$ to $|\mathcal{T}| = 6, 8, 10, 12$ and for each cardinality we consider 20 reduced instances. These instances are obtained from the competition ones by randomly selecting the teams to remove and by deleting them from all the constraints in which they appear. Table 5.6 (right) reports the same data as in Table 5.6 (left), except for the first column which, in this case, reports the size of set $\mathcal{T}$. As expected, the larger the size of $\mathcal{T}$, the less feasible (and optimal) solutions the solver is able to provide. Moreover, we observe that the model is solved consistently on instances up to size ten. Starting from $|\mathcal{T}| = 12$, the performances of the model drop drastically: only four feasible solutions and an optimal one are found with $|\mathcal{T}| = 12$ and only two feasible solutions are found with $|\mathcal{T}| = 14$.

In what follows, we discuss the results obtained by the Simulated Annealing algorithm on the competition instances. Specifically, we report both the best overall solution that we obtained for each instance and data on the average behavior of the algorithm. In order to assess the average performances of the Simulated Annealing in terms of cost, time and feasibility, we report the results of a dedicated experiment batch. All the experiments were run without a time limit, but rather with a fixed number of maximum iterations per stage (see Section 5.3.4). Depending on the number

Table 5.6: Results obtained solving Model $\mathcal{M}$ on the first (left) and second cluster (right) of reduced instances.

Removed constraints	# Inst.	# Feas.	# Opt.
CA	45	3	3
GA	42	1	0
BR	45	3	0
FA	23	0	0
SE	24	0	1
CA, GA	42	8	3
CA, BR	45	20	6
CA, FA	23	6	4
CA, SE	24	4	3
GA, BR	42	7	0
GA, FA	22	0	0
GA, SE	24	1	0
BR, FA	23	4	3
BR, SE	24	2	2
FA, SE	3	0	0

$\lvert\mathcal{T}\rvert$	# Inst.	# Feas.	# Opt.
6	20	7	13
8	20	14	6
10	20	15	2
12	20	4	1
14	20	2	0

Table 5.7: Iterations per stage, feature-based.

Instance type	Max iterations ($\mathcal{I}$) per stage		
	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$
$phased \wedge N_h > 200$	500000000	50000000	40000
$\neg phased \vee N_h \leq 200$	20000000	250000000	40000

of hard constraints in the instance, two different configurations of iterations per stage were applied, as shown in Table 5.7.

Table 5.8 reports the results obtained by the solver. The column *Best solution found* reports the best solution that our solver was able to find in all experiments. Some of these values are those that we submitted to the ITC2021 competition, others have been found in later experiments. When no feasible solution has been found, the number of hard violations followed by a letter H is reported. Next columns, labeled *Average values*, report the data obtained in a set of experiments that we run independently from the competition, in order to extract information on the average behavior of the algorithm in its final configuration. At least 48 runs per instance were performed to collect this data. Columns *Cost* and *Time* report, respectively, the average values of the objective function and the average time needed for a complete run of the three stages. Regarding the average cost, the value is computed only on feasible solutions. Column *Feasible* reports the

Table 5.8: Best and average results

Instance	Best solution found	Average values			Best known cost	CPLEX best bound
		Cost	Time (s)	Feasible		
Early_1	423	540.7	5667	1.00	362	1.0
Early_2	318	384.6	14844	1.00	145	0.0
Early_3	1068	1176.5	12195	1.00	992	48.9
Early_4	556	1007.8	8760	0.56	507	0.0
Early_5	4117	-	28517	0.00	3127	247.2
Early_6	3927	4543.0	35162	1.00	3325	587.3
Early_7	5205	6721.7	37487	1.00	4763	1233.1
Early_8	**1051**	1152.9	21394	1.00	1051	212.1
Early_9	132	228.7	10324	1.00	108	0.0
Early_10	4986	-	35856	0.00	3400	308.2
Early_11	4526	5784.5	43692	1.00	4426	309.5
Early_12	1010	1200.2	14726	1.00	380	0.0
Early_13	173	233.8	19675	1.00	121	2.0
Early_14	63	82.3	5616	1.00	4	1.0
Early_15	3556	3945.8	46715	1.00	3362	484.5
Middle_1	5657	6075.0	26291	0.06	5177	2857.5
Middle_2	5H	-	26891	0.00	7381	2909.8
Middle_3	**9542**	11403.1	44749	0.23	9542	3266.8
Middle_4	16	33.0	5660	1.00	7	7.0
Middle_5	510	624.4	6223	1.00	413	46.8
Middle_6	1701	2186.3	21350	1.00	1120	23.0
Middle_7	2203	2452.7	16303	1.00	1783	23.6
Middle_8	136	196.6	19718	1.00	129	2.0
Middle_9	640	772.1	17611	1.00	450	0.0
Middle_10	1357	1687.5	14433	1.00	1250	3.8
Middle_11	2696	2996.5	43877	1.00	2446	345.0
Middle_12	950	1054.2	14599	1.00	911	1.0
Middle_13	362	479.3	15687	1.00	252	0.0
Middle_14	**1172**	1304.6	37484	1.00	1172	0.0
Middle_15	985	1099.7	8705	1.00	485	0.7
Late_1	2021	2372.7	20242	1.00	1922	1102.4
Late_2	5715	6085.5	41433	0.49	5400	2817.7
Late_3	2457	2718.0	18328	1.00	2369	347.9
Late_4	**0**	0.0	2355	1.00	0	0.0
Late_5	2341	-	9191	0.00	1923	397.5
Late_6	930	1121.3	7122	1.00	923	5.4
Late_7	1765	2226.5	22959	1.00	1558	3.1
Late_8	997	1155.3	11286	1.00	934	77.9
Late_9	715	881.2	25963	1.00	563	2.9
Late_10	2571	3527.3	32511	0.05	1945	1.0
Late_11	207	289.3	15892	1.00	202	0.0
Late_12	3944	4830.6	35514	1.00	3428	156.2
Late_13	1868	2285.5	21007	1.00	1820	6.1
Late_14	**1202**	1326.3	39161	1.00	1202	6.5
Late_15	60	82.8	6435	1.00	20	0.0

ratio between feasible solutions and total runs. Finally, column *Best known cost* contains the best known results at the moment the article in Rosati et al. (2022) was written, according to data published on the website of the competition [3], and column *CPLEX best bound* contains the best lower bound that CPLEX was able to determine on Model $\mathcal{M}$. When the current best was determined by our solver, the corresponding value in the first column is marked in bold. Overall, our three-stage multi-neighborhood Simulated Annealing solver could find at least one feasible solution on 44 out of 45 instances. According to data, in its final configuration it manages to determine very easily a feasible solution on 36 instances, which are characterized by a feasibility ratio of 1.00, as it can be observed in column *Feasible* of the above-discussed table. The other instances appear to be less easy to solve for the algorithm. In particular, instances Early_5, Early_10, Middle_2, and Late_5 result to be considerably challenging, as feasible solutions are found just sporadically.

5.4.4 Analysis of PartialSwapTeamsPhased

One of the main contributions of the presented work is the new neighborhood PartialSwapTeamsPhased, that was introduced with the main purpose to solve the phased version of the problem. In order to test the effectiveness of the novel neighborhood, we run an additional set of experiments on phased instances without making use of PartialSwapTeamsPhased. To do so, we associated a probability of 0.00 to PartialSwapTeamsPhased in the multi-neighborhood and rescaled the probabilities associated to the other neighborhoods proportionally, in order to keep the same mutual ratios among them (according to the values in Table 5.4).

In Table 5.9 we report the average costs and the feasibility ratios obtained by the standard configuration and those obtained by the configuration that doesn't employ PartialSwapTeamsPhased. At least 20 runs per instance were executed. The last column reports, where possible, the percentage gap between the average cost obtained without PartialSwapTeamsPhased and the average cost obtained by the full configuration. It is possible to observe that instance Middle_1 is solved to feasibility only in the configuration that employs PartialSwapTeamsPhased. 16 instances, solved by both configurations, have worse results when solved without PartialSwapTeamsPhased. For just one instance, Late_4, the configurations obtain the same average cost. Finally, the remaining four instances, Early_5,

[3] `https://robinxval.ugent.be/ITC2021/`

Table 5.9: Comparison of the results obtained on phased instances with and without the neighborhood PartialSwapTeamsPhased.

Instance	With PSTP		Without PSTP		gap
	avg	feasible (%)	avg	feasible (%)	(%)
Early_1	540.7	1.00	563.5	1.00	+4.22%
Early_2	384.6	1.00	388.3	1.00	+0.96%
Early_3	1176.5	1.00	1204.4	1.00	+2.37%
Early_4	1007.8	0.56	1125.8	0.38	+11.71%
Early_5	-	0.00	-	0.00	-
Early_6	4543.0	1.00	4553.2	1.00	+0.22%
Early_10	-	0.00	-	0.00	-
Early_12	1200.2	1.00	1326.4	1.00	+10.51%
Middle_1	6075.0	0.06	-	0.00	$+\infty$
Middle_2	-	0.00	-	0.00	-
Middle_4	33.0	1.00	33.3	1.00	+0.91%
Middle_5	624.4	1.00	656.8	1.00	+5.19%
Middle_6	2186.3	1.00	2224.7	1.00	+1.76%
Middle_10	1687.5	1.00	1766.8	1.00	+4.70%
Middle_11	2996.5	1.00	3131.8	1.00	+4.52%
Middle_12	1054.2	1.00	1137.0	1.00	+7.85%
Late_4	0.0	1.00	0.0	1.00	+0.00%
Late_5	-	0.00	-	0.00	-
Late_6	1121.3	1.00	1141.7	1.00	+1.82%
Late_8	1155.3	1.00	1186.1	1.00	+2.67%
Late_10	3527.3	0.05	3590.0	0.05	+1.58%
Late_11	289.3	1.00	321.6	1.00	+11.16%

Early_10, Late_5 and Middle_2, are not solved to feasibility by any of the two configurations, in the given number of runs. According to these data, the neighborhood PartialSwapTeamsPhased appears to bring a tangible improvement in 17 out of 22 phased instances.

5.5 Comparison with other algorithms

A study realized after the ITC2021 competition compared the performance of eight algorithm (seven of them took part in the competition, including the winner) on an additional set of 518 instances and on the 45 instances of the competition, for a total of 563 instances[4]. The algorithms are listed in Table 5.10. All the algorithms were given two weeks of elapsed time to produce solutions for the whole instance set, but without specific time restrictions on the single runs and on the number of cores used. On the one hand, this might be regarded as a not fully fair comparison, given that different solvers had access to different computational resources (see

[4]The full study is available at Van Bulck et al. (2023).

Table 5.10: Overview of algorithms together with software and hardware details. The last column compares processor clock speeds relative to the 'fastest' processor used (3.9 GHz).

Algorithm	Search method	Software details	Clock speed ratio	Reference
MODAL	IP Branch & Cut	Python, Zimpl, C, Gurobi 10, Xpress	2.8/3.9	Berthold et al. (2021)
Goal	Fix-and-optimize matheuristic	Java 16, Gurobi 10.0	3.2/3.9	Fonseca and Toffolo (2022)
DES	Adaptive LNS matheuristic	Python 3.10, Gurobi 10.0	3.9/3.9	Phillips et al. (2021)
UoS	VND matheuristic	Python 3.10.4, Gurobi 9.0.2	2.0/3.9	Lamas-Fernandez et al. (2021)
Udine	Simulated annealing	C++17	2.4/3.9	Rosati et al. (2022)
DITUoIArta	CP/SAT + Simulated annealing	Python 3.10, ORTools 9.4	3.2/3.9	Dimitsas et al. (2022)
Reprobate	Pseudoboolean optimization	Perl, clasp 3.3.9, Sat4J 2.3.6, RoundingSat	3.2/3.9	Lester (2022)
FBHS	IP Decomposition + matheuristic	C++, CPLEX 12.10	2.6/3.9	Van Bulck and Goossens (2023a)

Fig. 5.10, that provides an overview of the CPU times used by the algorithms) and not all solvers were written in the same language and with the same technologies (see Table 5.10). For example, C++, that is a compiled language, produces much faster code than Python, that runs through an interpreter. Nonetheless, sports timetables are planned on a yearly basis and have a relevant impact on the attendance and profitability of the competition, making running time a secondary aspect. Moreover, these solvers were designed and developed under a particular circumstance (the ITC2021 competition), and comparing them under exactly equal conditions would imply a relevant effort in terms of time and resources. For this reason, we consider these results as the best possible comparison that can be realized at the moment.

Defined $\Delta_{i,a}$ as the difference between the best solution $F_{i,a}$ found by algorithm a on instance i and the best performance achieved across all algorithms on instance i (i.e., $\Delta_{i,a} = F_{i,a} - \min_a F_{i,a}$), Fig. 5.11 plots the distribution of $\Delta_{i,a}$ among these problem instances and the number of problem instances for which a found a feasible solution. The figure shows

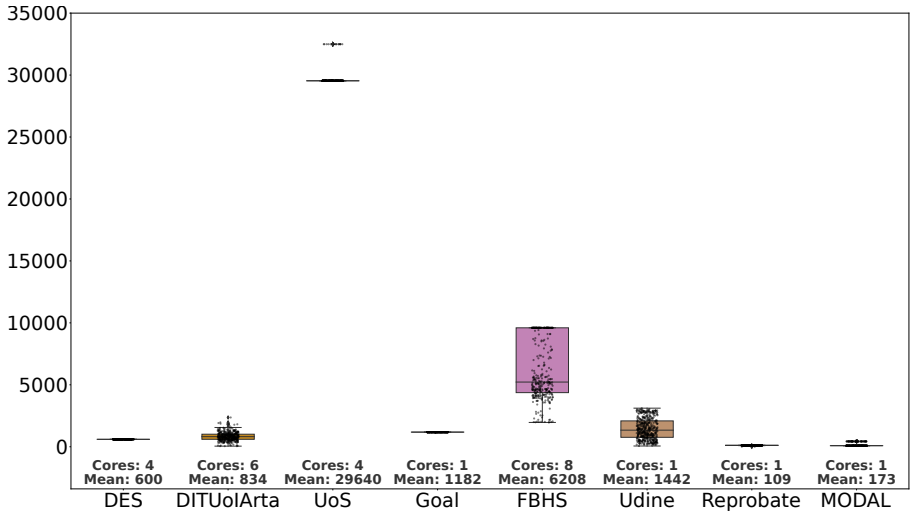
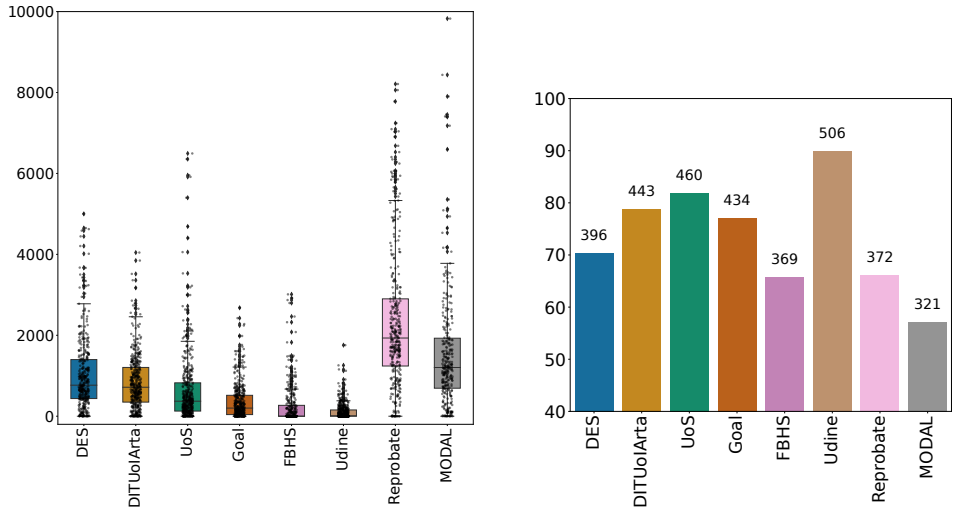


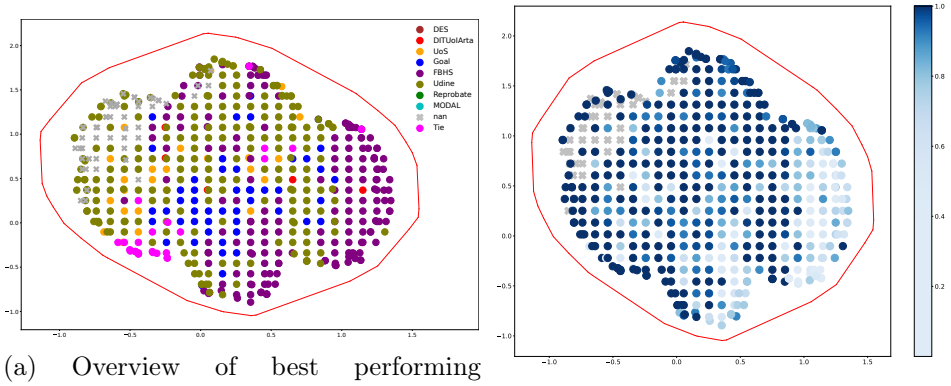
Figure 5.10: Distribution of the CPU runtimes for the additional problem instances per algorithm.



(a) $\Delta_{i,a}$ (b) Feasible solutions

Figure 5.11: Distribution of $\Delta_{i,a}$ and number of instances solved to feasibility.

that our algorithm managed to find a feasible solution for a large number of instances, 508 out of 563, more than any other solver, and that these solutions are never far from the best found solutions.



(a) Overview of best performing algorithm on the feature space

(b) Regret Udine

Figure 5.12: Algorithm performance on the feature space

Additionally, Fig. 5.12a shows the winning algorithm in terms of the best solution found for each of the problem instances, plotted on a bidimensional plane obtained from a principal component analysis (PCA) of the features of the instances (for more details, see Van Bulck and Goossens, 2023b). The color denotes the best algorithm for a given point, problem instances for which none of the algorithms found a solution are indicated by a gray x-mark. From the figure we can see that most of the algorithms perform particularly well in a specific part of the feature space and no single algorithm dominate all the feature space.

In Fig. 5.12b, that uses a non-linear color scheme to superimpose our relative gap $\Delta_{i,a}$ in the 2D feature space, we highlight with a darker color the instances where our solution method performs exceptionally well. It shows the algorithm effectiveness in the part of the feature space where phased problem instances are projected, witnessing the contribution of the newly proposed PartialSwapTeamsPhased neighborhood. Only near the origin of the feature space, there is a gap with Goal and UoS, and in the far right of the feature space, the FBHS solver performs better.

This comparison suggests that our Multi-Neighborhood Search approach works better than the other solvers both in terms of feasibility and solution quality, and is a top performer on a large portion of the feature space.

5.6 Conclusions

In this study, we considered the version of the Sports timetabling Problem proposed for the ITC2021 competition. We presented an ILP model for

the problem, which did not yield significant results when solved by a commercial solver. Then, we tackled the problem employing a three-stage Multi-Neighborhood Simulated Annealing approach, that makes use of six different neighborhoods. In particular, the neighborhood that we named PartialSwapTeamsPhased is a novel contribution.

This approach managed to find a feasible solution for 44 out of the 45 instances proposed by the competition. Feasible solutions were found rather easily for most of the instances, however the metaheuristic struggled to produce feasible solutions for certain instances, even in long execution times. The results obtained by the Simulated Annealing approach allowed us to rank second out of 13 participants in the final ranking of the competition.

Furthermore, a comparative analysis of our solution method against other methods across a dataset consisting of 563 instances, that includes the 45 competition instances, revealed that our algorithm yields the best results with regard to both feasibility and solution quality.

Chapter 6

Home Healthcare Routing and Scheduling

Over the past two decades, we have witnessed a shift in the provision of supportive and geriatric care. Instead of being primarily based in institutional settings like nursing or rest homes, these services have been increasingly relocated to the patients' own homes.

The reasons behind this shift can be attributed to two factors. Firstly, allowing patients to remain in their familiar environments enhances their quality of life. Secondly, this situation has a notable positive effect on reducing healthcare costs, a sensible topic in rapidly aging societies.

Trained caregivers provide home healthcare services by visiting the patient's home during a designated time window. They carry out service operations based on the patient's specific needs, which can range from medical care to instrumental activities of daily living. After completing their tasks, the caregivers proceed to the next patient. This distinct characteristic of home healthcare makes it a structured problem. Unlike classical activity scheduling problems in hospitals or healthcare institutions, which primarily focus on temporal aspects, home healthcare requires consideration of both temporal and spatial dimensions, including travel times between patients.

The synchronization constraint stands out as the most distinctive type of constraint in this problem, as it introduces temporal dependencies among the activities. For instance, certain medical care tasks such as physiotherapy necessitate the simultaneous presence of multiple caregivers, such as when lifting the patient from their bed. Other activities, such as administering medication or preparing lunch, may require subsequent actions to take place after a specific time interval, like administering one dose in the morning

and another in the evening. Due to the non-negligible travel time involved when caregivers transition from one patient to another, these constraints significantly increase the complexity of the routing aspect of the problem compared to standard vehicle routing problems.

In this chapter, we present a Multi-Neighborhood Search approach to solve the problem formulation introduced by Mankowska et al. (2014). Our method incorporates a diverse set of original neighborhood operators. The core of the solution process relies on the implementation of a Simulated Annealing (SA) procedure (Kirkpatrick et al., 1983) for driving the search.

The approach outperforms previous methods in the literature (Mankowska et al., 2014; Lasfargeas et al., 2019; Kummer et al., 2020; Kummer, 2021; Kummer et al., 2022) and shows its capability to improve the current state-of-the-art results on the benchmark instances.

In addition, we have designed a parametric instance generator that relies on real geographical data and real population density. This generator allows us to create an additional dataset that encompasses a significantly wider range of sizes and features' values compared to the existing benchmarks proposed by Mankowska et al. (2014) and Kummer (2021). For our new dataset, we have introduced a novel file format based on JSON, which offers enhanced robustness, extensibility, and human readability when compared to the previous formats utilized thus far. We also show our results on the new instances, highlighting the relationship between the different objectives.

6.1 Related work

Research on scheduling of home healthcare services has been ongoing since the late 1990s. Notably, Begur et al. (1997) were among the first to consider this problem and applied a simple scheduling heuristic as its solution method. Subsequently, Cheng and Rich (1998) formulated the problem using a mixed-integer linear programming techniques.

Later attempts to address this problem approached it from a *set covering* perspective. For instance, Eveborn et al. (2006) introduced a working healthcare planning system, while Rasmussen et al. (2012) were pioneers in considering temporal dependencies among the activities. Bredström and Rönnqvist (2008) also worked on a similar synchronization problem but looking at it from a vehicle routing perspective, without explicitly considering the healthcare nature of the services involved.

The idea of providing services in a metropolitan area, where the travel between patients can be facilitated using a public transportation network

instead of a private car, has been explored in the studies conducted by Bertels and Fahle (2006) and Rendl et al. (2012). The latter research also considers the option of switching the mode of transportation during the journey, resulting in multi-modal trips.

Di Gaspero and Urli (2014) tackled a similar problem, albeit with a distinct perspective. Their study focused on a temporal horizon that extended over multiple days, and they also incorporated the balancing of caregivers' workloads, aiming to minimize overtime. Additionally, in their approach, there was room to leave certain patient activities unscheduled, which proved helpful in handling overconstrained scenarios and allowed for the possibility of hiring more occasional caregivers if needed. To address this problem, they model it in the Constraint Programming framework and employed specialized branching heuristics and a Large Neighborhood Search approach for the solution.

Mankowska et al. (2014) presented a relatively standardized representation of the problem and also offered a set of benchmark instances for evaluation purposes. The problem formulation (refer to Section 6.2 for an informal overview) incorporates synchronization constraints at patients' homes, allowing for a maximum of two activities to be coordinated. The quality of the solution is assessed based on traveling times and the tardiness of service activities in relation to patients' time windows. To solve this problem, Mankowska et al. devised an Adaptive Variable Neighborhood Search method that incorporates eight distinct neighborhood operators.

A similar problem was addressed by Ait Haddadene et al. (2016), but in their formulation patients' time windows must be strictly respected and the objective is to minimize total traveling times and patients' non preferences related to caregivers. A MIP model and a hybrid Greedy Randomized Adaptive Search Procedure and Iterated Local Search metaheuristic are proposed and compared on the testbed designed by Bredström and Rönnqvist (2008), conveniently extended to take into account different types of services.

Recent advancements in the field include the research conducted by Lasfargeas et al. (2019). They proposed a local search-based method integrated into a Variable Neighborhood Search solution procedure and evaluated it using the Mankowska et al. (2014) benchmarks. Population-based approaches have also been explored by various researchers, tailored to different problem settings. For instance, Decerle et al. (2018) and Grenouilleau et al. (2019) developed memetic algorithms, which involve genetic algorithms followed by a local search step, to address the problem. On the other hand, Clapper et al. (2023) introduced a

model-based evolutionary algorithm. Following a similar approach to the work by Di Gaspero and Urli (2014), Grenouilleau et al. (2019) considered a multi-day horizon and aimed to balance caregivers' workload, including minimizing overtime. However, their routing problem was more sophisticated, accounting for hourly dependent traveling times due to traffic. Moreover, Xiang et al. (2021) and Oladzad-Abbasabady et al. (2023) approached the problem from a multi-objective perspective, seeking to achieve a balance between the total operating cost and the satisfaction of caregivers and patients. They employed NSGA-II genetic and Iterated Local Search algorithms, respectively, to tackle the problem.

The problem formulation introduced by Mankowska et al. (2014) has attracted interest from Kummer and co-workers (Kummer et al., 2020; Kummer, 2021; Kummer et al., 2022), who developed and utilized Biased Random Key Genetic Algorithms to address the problem. Their outcomes currently stand as the state-of-the-art concerning these benchmarks. Additionally, in their work, Kummer (2021) introduced a new dataset that includes a more comprehensive set of features compared to the original dataset proposed by Mankowska et al. (2014).

Recently, the home health care routing and scheduling problem with stochastic travel and service times was examined by Bazirha et al. (2023). They formulated the problem as a two-stage stochastic programming model with recourse, which involves penalty costs for delayed services to patients and remuneration for caregivers' extra working time. To solve the deterministic model, the researchers employed CPLEX, a genetic algorithm, and variable neighborhood search-based heuristics. For the stochastic programming model, they utilized Monte Carlo simulation embedded into the genetic algorithm. The performance of these solution methods was evaluated on test instances generated in accordance with the benchmark instances proposed by Mankowska et al. (2014).

Notable surveys on the utilization of operational research methods in the context of home healthcare problems are presented in the works of Fikar and Hirsch (2017), Cissé et al. (2017) and Grieco et al. (2021).

6.2 Problem definition

The Home Health Care Routing and Scheduling Problem (HHCRSP) has been formally defined by Mankowska et al. (2014) using a mathematical model. For the sake of completeness, we present a summary of its definition in this chapter.

The primary entities involved in the problem are as follows:

Times and Horizon: The problem consists in planning for a single day, with time expressed in minutes, starting from 0, which corresponds to the beginning of daily activities (e.g., 6:00 AM) when all caregivers are assumed to be at the *central office*. There is no explicit time horizon, and the activities are limited by the patient's time windows. Distances, both between patients and from the central office, are directly measured in minutes required to travel and cover those distances.

Patients: Patients are categorized into two classes: *single-service* patients and *double-service* ones. Single-service patients require service from one caregiver within their designated time window. In contrast, double-service patients need to be served by two caregivers, either *simultaneously* or in a *sequential* manner, within their time window. For sequential double-service patients, the minimum and maximum time gap between the two services is explicitly specified.

Services and Caregivers: Every service has its own duration, which may differ based on the individual patient's requirements. Each caregiver is specialized in providing a particular service, leveraging their specific abilities. The caregivers start their daily tasks from the central office and complete their workday by returning to the same central location.

According to the model of Mankowska et al., service time can vary depending on the specific service and the patient, even though in the original dataset this length is a global constant value. In the dataset of Kummer (2021), the values actually differ between services and patients, although they do not depend on the specific caregiver delivering the service. In the construction of our dataset, we also assume that the services provided to patients depend on the type of service and the patient, rather than the caregiver themselves.

Fig. 6.1 shows the data of a toy instance with six patients (p_1, p_2, ..., p_6), three caregivers (c_1, c_2, and c_3), and three services (s_1, s_2, and s_3). As mentioned above, distances, service times, and time windows are expressed in minutes. For example, assuming that the working day starts at 6:00AM, the first time window 240 – 360 represents the fact that patient p_1 should be visited between 10:00AM and 12:00AM. Furthermore, p_1 needs a single service s_2, whose duration is 30 minutes. Conversely, patients p_4 and p_5 require a double service: specifically, p_4 needs the simultaneous presence of two caregivers, whereas for p_5 the two services must be separated by at least

Patients

ID	Time window	Service (duration) first	second	Separation min	max
p_1	240 – 360	s_2 (30)	—		—
p_2	120 – 180	s_3 (20)	—		—
p_3	0 – 60	s_2 (45)	—		—
p_4	120 – 210	s_2 (30)	s_3 (30)	0	0
p_5	270 – 420	s_1 (15)	s_3 (30)	30	45
p_6	360 – 420	s_1 (45)	s_3 (20)	60	90

Caregivers

ID	Services
c_1	s_1, s_2
c_2	s_3
c_3	s_2, s_3

Distances

co	0	38	34	56	7	13	26
p_1	39	0	23	22	32	50	58
p_2	35	23	0	44	28	47	43
p_3	56	22	44	0	54	59	78
p_4	7	32	28	51	0	19	28
p_5	13	45	47	59	19	0	35
p_6	27	57	42	77	28	35	0
	co	p_1	p_2	p_3	p_4	p_5	p_6

Figure 6.1: Toy instance data

30 minutes and at most 45 minutes. The central office is denoted as *co* in the distance matrix. Notice that distances are not symmetric.

The possibility that a single caregiver provides both services for the same double-service patient is obviously impossible for simultaneous services. It is also explicitly forbidden for sequential ones by the work by Mankowska et al.. From a practical standpoint, it may not be intuitive to impose this limitation, as using a single caregiver has the potential to save time in the overall schedule. However, in real-life scenarios, it is assumed that the two services (e.g., nursing and physiotherapy) require distinct skills and qualifications. Therefore, it is highly improbable that a single caregiver possesses both sets of skills. In the available datasets, the case that a single caregiver has the ability for two sequential services at the same patient never occurs, so this is not an option. We keep the same limitation and this situation does not occur in our dataset either[1].

[1]Notice that the toy instance, due to the limited size, does not enforce this limitation.

Enforcing the caregiver's qualification for the specific service is a hard constraint. Likewise, it is strictly prohibited for a service to commence before a patient's designated time window begins. Following standard practices in vehicle routing, if the caregiver arrives early, it is required to wait until the patient's time window starts. On the other hand, the occurrence of a patient being served late is admissable, and the extent of tardiness is considered in the objective function. Finally, the min and max time separation between sequential services for a double-service patient is also a hard constraint.

Notice that it is also possible that some caregivers are not assigned to any patient, thus they have an empty route. Conversely, all patients must be covered for all the services they need.

The problem goal is to minimize an objective function that consists of three components: the total travel time, the overall tardiness, and the highest individual tardiness. The inclusion of the highest tardiness component is essential to ensure fairness among patients. This prevents the best solution from being achieved at the expenses of a single patient, solely by significantly delaying his/her service provision. In cases where a patient receives double service, each service tardiness is considered independently. All three components are measured in minutes and are combined with equal weight, each contributing $\frac{1}{3}$ to the overall objective.

Fig. 6.2 shows a (non-optimal) solution to the instance of Fig. 6.1, in the visual form automatically produced by our solution validator. The time windows of patients are highlighted in dark gray and the warehouse icon refers to the central office. The travel times of caregivers c_1, c_2, and c_3 are 190 (56 + 22 + 50 + 35 + 27), 39 (7 + 19 + 13), and 117 (34 + 28 + 28 + 27), respectively. Notice that all services are provided on time, except for service s_3 by caregiver c_3 at patient p_6 (shown with white diagonal stripes in the figure) that is late by 10 minutes. Hence, the total and highest lateness values are both 10. Consequently, the cost of this solution is calculated as (190 + 39 + 117)/3 + 10/3 + 10/3, resulting in a total cost of 122. Notice also that there are three early arrivals, with long waiting times (shown in dotted rectangles in the figure). This phenomenon also happens in other instances, but it is particularly evident in this toy instance.

6.3 Solution method

We present a Multi-Neighborhood Search approach to address the problem. In the subsequent sections, we will outline the fundamental components of the local search paradigm, which include the *search space*, the *initial solution*

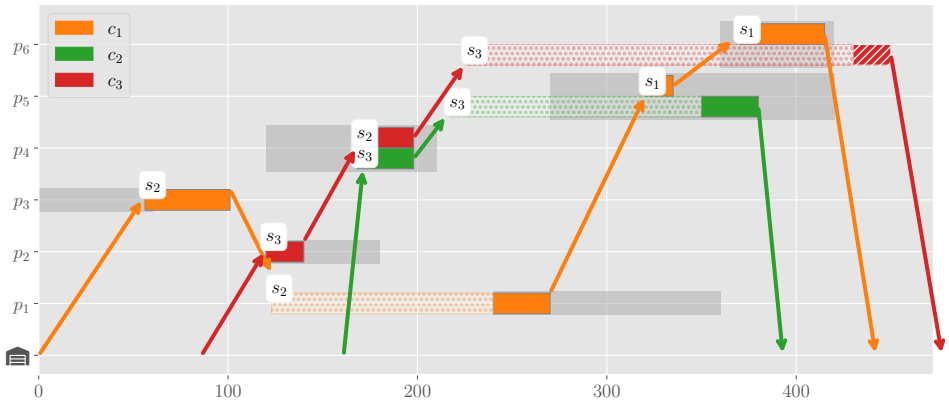


Figure 6.2: A solution of the toy instance

strategy, the *neighborhood relations*, and the *metaheuristic* that drives the search process.

In the following, we denote by $\mathcal{P} = \{1, \ldots, P\}$ the set of patients, $\mathcal{C} = \{1, \ldots, C\}$ the set of caregivers, and $\mathcal{S} = \{1, \ldots, S\}$ the set of services.

6.3.1 Search space

Following Mankowska et al. (2014), the search space is represented by a vector $\Pi = [\pi_1, \pi_2, \ldots, \pi_P]$ containing a permutation of the values $1, \ldots, P$, which represents a global ordering of the patients. That is, patient π_1 is the first patient to be served, π_2 is the second, and so on.

Vector Π is completed by another P-sized vector of pairs $\Theta = [(\vartheta_{1,1}, \vartheta_{1,2}), (\vartheta_{2,1}, \vartheta_{2,2}), \ldots, (\vartheta_{P,1}, \vartheta_{P,2})]$ such that $\vartheta_{p,1}$ and $\vartheta_{p,2}$ are the caregivers that serve patient p. In case of a single-service patient p, the second element of the pair $(\vartheta_{p,2})$ is not used.

For example, the solution of Fig. 6.2 to the instance of Fig. 6.1 is represented by the vectors: $\Pi = [3, 2, 4, 1, 5, 6]$ and $\Theta = [(1, -), (3, -), (1, -), (3, 2), (1, 2), (1, 3)]$, where the dash symbol $-$ means that the value is not present.

The routes of caregivers and their corresponding service times are deterministically constructed, starting from sets Π and Θ (see Section 2.2.5). The *scheduling procedure* starts with empty routes and processes patients one by one, following the order specified in Π. During each iteration i of this procedure, patient $p = \pi_i$ is added to the end of caregivers' routes $c_1 = \vartheta_{p,1}$ and $c_2 = \vartheta_{p,2}$ (or only to c_1 for patients requiring a single service).

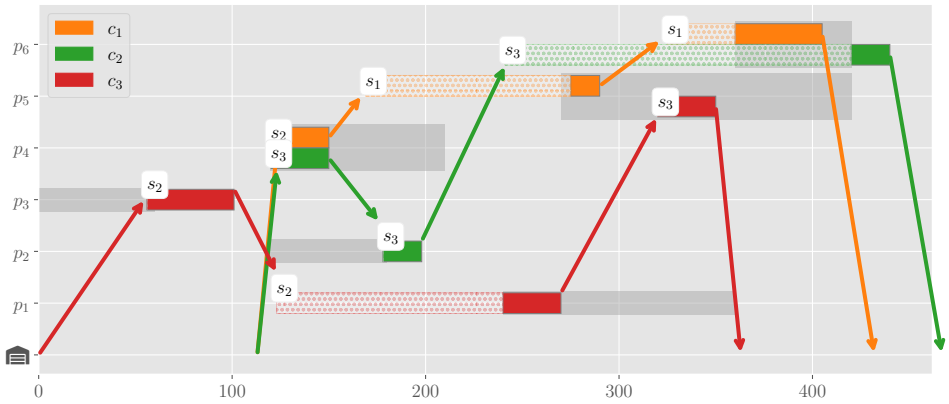


Figure 6.3: The optimal solution of the toy instance

The service start times are calculated *at the earliest*, taking into consideration the time window of patient p. Specifically, for caregiver c_1 at patient p, the start time is determined as the maximum value between the beginning of the time window for p and the earliest time at which c_1 can reach p based on their previous assigned duties and travel times. For the second service (if applicable), the service start time of caregiver c_2 at patient p is determined as the maximum value between the minimum separation between the two services for p and the earliest time at which c_2 can reach the patient.

The solution of Fig. 6.2 is obtained applying the scheduling procedure. We can see that all services are provided always as soon as possible.

In the case in which c_2 reaches patient p later than the maximum separation between the two services of p, the service of c_1 is postponed as much as necessary in order to keep the separation within the range.

This case happens for the solution of the toy instance represented by $\Pi = [4, 3, 1, 5, 2, 6]$ and $\Theta = [(3, -), (2, -), (3, -), (1, 2), (1, 3), (1, 2)]$, which incidentally is optimal, with no tardiness, and a total traveling time of 334 minutes, corresponding to a cost of 111.33. The schedule built for this solution by the above procedure is shown in Fig. 6.3, where we notice that caregiver c_1 (orange) at patient p_5 is postponed by 5 minutes to time 275 from the beginning of the time window at 270, because caregiver c_3 can arrive at the patient only at time 320 and the maximum separation is 45 minutes. Notice also that there is no tardiness as long as the caregiver starts within the time window, regardless of the fact that the service finishes after the end of the time window (see caregiver c_2 at patient p_6).

With this scheduling procedure, separations are always satisfied, although tardiness is possible. By treating tardiness as a soft constraint, this ensures that all necessary services can be accommodated within the schedule, guaranteeing the feasibility of the solution.

The two vectors Π and Θ are sufficient to represent the solution and to build the full schedule by using the procedure described above. Nonetheless, in our implementation they are complemented by many redundant data structures, used to accelerate the evaluation of the costs of neighbor solutions. These data structures include, among others, the position of each patient in the ordering (the inverse of Π), the route of each caregiver, and the positions of the patients in the routes of their caregivers.

6.3.2 Initial solution

The initial solution is created randomly. Initially, a random permutation is chosen for the Π vector. Next, one (or two distinct) caregivers are randomly selected for each patient and added to the Θ vector. The caregivers are chosen from those who possess the required ability to provide the service.

Given the vectors Π and Θ, the scheduling procedure is applied, so as to produce the full initial schedule and to evaluate corresponding costs.

6.3.3 Multi-neighborhood

We consider three neighborhood relations: MovePatient, which repositions and reassigns one patient; SwapPatients, which swaps two patients in terms of global position and caregivers; and InRouteSwap, which swaps two patients within a specific route.

The set of neighborhoods we employ has a significantly broader scope compared to the one utilized by Mankowska et al. (2014). Our neighborhoods enable simultaneous changes in both the position and caregivers, whereas Mankowska et al. considered only the options of repositioning a patient within the global order, changing the caregiver(s), *or* swapping either the position or caregiver(s) individually.

For the sake of clarity, we illustrate the neighborhoods only for the case of double-service patients, which is the most complex one. The case of single-service ones is obtained simply by ignoring the second caregiver and adjusting the various features accordingly.

Furthermore, throughout this section, in order to simplify the presentation, given a solution and a patient p, we use the following notation. We call c_1^p and c_2^p the caregivers assigned to p, i.e., $c_1^p = \vartheta_{p,1}$, $c_2^p = \vartheta_{p,2}$; we

call i^p the current position of p in Π; finally, we call s_1^p and s_2^p the services requested by patient p.

Neighborhood MovePatient

The first neighborhood, called MovePatient (MP), consists in repositioning one patient p in the global ordering *and* assigning new caregivers to p, in one single move:

- **Attributes**: $\langle p, i, c_1, c_2 \rangle$, with $p \in \mathcal{P}$, $1 \leq i \leq P$, $c_1, c_2 \in \mathcal{C}$
- **Preconditions**:
  - No null moves: $\langle i, c_1, c_2 \rangle \neq \langle i^p, c_1^p, c_2^p \rangle$
  - Distinct caregivers: $c_1 \neq c_2$
  - No missing abilities: c_1 and c_2 have the ability for the services
- **Effects**: Patient p is moved to position i in the global ordering Π. All patients in positions between i and i_p are shifted accordingly (either forward or backward). Caregivers c_1 and c_2 are assigned to p. The full schedule is recomputed using the defined procedure.
- **Special cases**: The position i can be the same as i_p, which means the move results in a change of caregivers only. Likewise, it is possible for one or both caregivers to stay unchanged, indicating that the move represents only a modification in the sequence of the routes.

As an example, consider the state in Fig. 6.3, represented by $\Pi = [4, 3, 1, 5, 2, 6]$ and $\Theta = [(3, -), (2, -), (3, -), (1, 2), (1, 3), (1, 2)]$. Now, take the move $\mathsf{MP}\langle 5, 2, 2, 3 \rangle$, which relocates patient 5 to position 2 and assigns caregivers 2 and 3. The application of this move to the given state would lead to the new state $\Pi = [4, \mathbf{5}, 3, 1, 2, 6]$ and $\Theta = [(3, -), (2, -), (3, -), (1, 2), (\mathbf{2}, \mathbf{3}), (1, 2)]$, where the affected values are highlighted in boldface. Notice that the second caregiver (3) is left unchanged.

Neighborhood SwapPatients

The second neighborhood, called SwapPatients (SP), consists in swapping the position in Π and the caregivers in Θ for two patients. A swap is possible only between patients with the same number of services and with current caregivers with the required abilities for the other patient.

For double-service patients, the neighborhood also includes the option that first and second caregivers are crossed between the two patients: the first caregiver of one patient is assigned to the second service of the other patient. This option is stored in a Boolean attribute of the move, called CS (for *cross swap*), such that if CS = F the caregivers are swaped position-wise, whereas if CS = T they are inverted. The definition of the neighborhood is the following:

- **Attributes**: $\langle p_1, p_2, CS \rangle$, with $p_1, p_2 \in \mathcal{P}$, $CS \in \{\mathsf{F}, \mathsf{T}\}$
- **Preconditions**:
  - No null moves: $p_1 \neq p_2$
  - Same type: p_1 and p_2 are both double-service or both single-service
  - No missing ability $c_1^{p_1}$, $c_1^{p_2}$, $c_2^{p_1}$, and $c_2^{p_2}$ have the ability for the services assigned to them by the move
- **Effects**: patient p_1 is moved in position i_2^p and patient p_2 is moved in position i_1^p; if $CS = \mathsf{T}$, then $c_1^{p_2}$ and $c_2^{p_2}$ are assigned to p_1, and $c_1^{p_1}$ and $c_2^{p_1}$ are assigned to p_2, for the first and second service, respectively; if $CS = \mathsf{F}$, then $c_2^{p_2}$ and $c_1^{p_2}$ are assigned to p_1 and $c_2^{p_1}$ and $c_1^{p_1}$ are assigned to p_2, for the first and second service, respectively.

Consider again the solution in Fig. 6.3, represented by $\Pi = [4, 3, 1, 5, 2, 6]$ and $\Theta = [(3, -), (2, -), (3, -), (1, 2), (1, 3), (1, 2)]$. The move $\mathsf{SP}\langle 5, 6, \mathsf{F} \rangle$ would lead to the state $\Pi = [4, 3, 1, \mathbf{5}, 2, \mathbf{6}]$ and $\Theta = [(3, -), (2, -), (3, -), (1, 2), (\mathbf{1}, \mathbf{2}), (\mathbf{1}, \mathbf{3})]$, where the affected values are highlighted in boldface.

Neighborhood InRouteSwap

The third neighborhood, called InRouteSwap (IRS), is also a swap move, but of a different type. It swaps two patients in positions j_1 and j_2 within the route of a given caregiver c. If one or both patients are double-service ones, the route of the *side* caregiver(s) serving the patient(s) is modified accordingly. That is, the patient is moved to the position in the route corresponding to their new global position. Differently from the SP neighborhood, side caregivers are not swapped, thus creating a different type of movement.

We call l_c the length of the route for caregiver c, i.e., the number of patients served by caregiver c. We also call p_1 and p_2 the patients in positions j_1 and j_2 of the route of c, and i_1 and i_2 their global positions, i.e., their positions in Π.

- **Attributes**: $\langle c, j_1, j_2 \rangle$, with $c \in \mathcal{C}$, $1 \leq j_1 < j_2 \leq l_c$.
- **Preconditions**:
  - Minimum route length: $l_c \geq 2$
- **Effects**: Patient p_1 is moved to global position i_1 and p_2 is moved to global position i_2. The routes of c and of the other caregivers of patients p_1 and p_2 are updated accordingly.

Consider once again the solution in Fig. 6.3, represented by $\Pi = [4, 3, 1, 5, 2, 6]$ and $\Theta = [(3, -), (2, -), (3, -), (1, 2), (1, 3), (1, 2)]$. The move $\mathsf{IRS}\langle 3, 2, 3 \rangle$ swaps the second and the third patients in the route of c_3. The new ordering obtained is $\Pi = [4, 3, \mathbf{5}, 1, 2, 6]$ (where the affected value is highlighted in boldface as before), Θ is unchanged. Notice that p_5 is a double-service patient who is also served by c_1, so the route of c_1 could be affected. In this case, however, given that the new position of p_5 is remained within the positions of p_4 and p_6, the route of c_1 is unchanged.

Composition of neighborhoods

The multi-neighborhood used in our solution method is $\mathsf{MP} \cup \mathsf{SP} \cup \mathsf{IRS}$, i.e., the set union of the three basic ones defined above. In particular, a random move from the union neighborhood is obtained by first selecting one of the three basic ones and then drawing a random move within that basic neighborhood.

The first selection is a weighted randomization that makes use of two parameters, called σ_{SP} and σ_{IRS}, so moves of type SP and IRS are drawn with probability σ_{SP} and σ_{IRS}, respectively. Consequently, we draw a MP move with probability $1 - \sigma_{\mathsf{SP}} - \sigma_{\mathsf{IRS}}$. Within the single neighborhood, the selection of the specific move is done uniformly.

It is worth mentioning that the MovePatient neighborhood is the most important one, as confirmed later by the experimental analysis. In fact, it is easy to verify that the search space is connected under this neighborhood, as it is possible to go from any solution to any other one, by means of a chain of MovePatient moves. This is not the case for the other two neighborhoods that

do not change the length of the route. They can be considered as auxiliary neighborhoods that allow to diversify the search.

We also notice that there is an overlap between SwapPatients and InRouteSwap, as a move that swaps two single service patients in the same route can be obtained from both of them. Given that we do not explore the neighborhood exhaustively, but rather sample random moves this is not a source of inefficiency, but only a (small) bias in the distribution.

6.3.4 Metaheuristic

To guide the local search, we employ the Multi-Neighborhood Simulated Annealing described in Chapter 2.

6.4 Datasets and generators

In this section, we first describe the datasets already available for this problem. Subsequently, we introduce our generator and we describe our new dataset obtained using the generator. Finally, we discuss the file formats for instances and solutions.

6.4.1 Dataset by Mankowska et al.

The instances of Mankowska et al. (2014) are artificial and created by a generator which samples random locations in a square of size 100×100 and assigns Euclidean (and thus symmetric) distances among them. The number of double service patients is fixed so as to be around 30% of the total number of patients, equally distributed between those requiring simultaneous and sequential services. The number of services is fixed globally to the value 6, and their duration is a random number between 10 and 20, and it is identical for all patients/caregivers within the instance.

The dataset generated consists of seven groups, each containing ten instances with varying patient counts: 10, 25, 50, 75, 100, 200, and 300 patients, respectively. These groups are denoted by the letters from A to G, resulting in instance names such as $A_1, A_2, \ldots, G_{10}$.

The upper section of Table 6.1 displays the primary characteristics of this dataset, encompassing the number of patients (P), services (S), caregivers (C), the percentage of double-service patients, the duration of time windows, and the minimum and maximum average compatibility between patients and caregivers (per service). Moreover, it includes the minimum and maximum

Table 6.1: Summary of features of the different instance groups.

Group	P	S	C	Double service	Time windows	Compatibility min–max	Distance min–max
Mankowska et al. (2014)							
A	10	6	3	30%	120	0.33–0.56	39.6–50.1
B	25	6	5	32%	120	0.35–0.52	38.4–45.9
C	50	6	10	30%	120	0.34–0.49	39.5–45.0
D	75	6	15	31%	120	0.33–0.46	39.3–44.7
E	100	6	20	30%	120	0.37–0.43	39.4–44.7
F	200	6	30	30%	120	0.38–0.46	38.9–42.0
G	300	6	40	33%	120	0.37–0.42	39.2–42.1
Kummer (2021)							
I–10	10	6	3	40%	120	0.33–0.67	7.8–27.6
I–25	25	6	5	32%	120	0.20–0.60	15.2–23.2
I–50	50	6	10	32%	120	0.10–0.50	11.9–24.9
I–75	75	6	15	32%	120	0.07–0.47	10.9–24.8
I–100	100	6	20	32%	120	0.10–0.50	11.4–22.6
I–200	200	6	40	30%	120	0.20–0.48	12.7–22.4
I–300	300	6	60	30%	120	0.22–0.43	12.3–22.1
I–400	400	6	80	30%	120	0.24–0.43	11.1–22.4

average distances in minutes between patients and also between patients and the central office.

It is worth noting that instances within different groups exhibit considerable homogeneity. This uniformity stems from the fact that they were generated using the same configuration of the generator.

6.4.2 Dataset by Kummer

Kummer (2021) made an analysis of the instances by Mankowska et al. (2014), highlighting that they are rather unrealistic, due to the lack of "structure" in the geographical data. For this reason, he developed a new generator that overcomes this weakness by sampling points in real cities and matching them with real addresses and computing real routes. As a consequence, distances are not Euclidean and not symmetric. However, the ratio between different patient types is fixed exactly as in the dataset by Mankowska et al. (2014),and the number of services is fixed to 6 as well. Nonetheless, the distribution of the abilities is more balanced in order to provide against the possibility that some services are assigned to a very small number of caregivers.

For the generation of the geographical data of the instances, Kummer identified three features, namely the node generation strategy (random or clustered), the central office placement strategy, and the cluster density. He selected 22 different combinations of values for these features and generated 100 instances for each combination. The values of the size of the groups are the same seven of Mankowska et al., plus an additional size of 400 patients, for a total of eight groups.

As a result, he generated 17,600 ($22 \times 100 \times 8$) instances that are publicly available at `https://github.com/afkummer/hhcrsp-dataset-2021` and can be used for training. The main features of this dataset are shown in the lower part of Table 6.1.

Inside this dataset, the author identified 160 instances (20 for each group) as the validation ones. Within each group, the set of 20 instances is composed of the 10 hardest ones and 10 random ones. The hardness of an instance is measured as the different between the lower bound and the best obtained value within a given time limit. The lower bounds are computed relaxing synchronization constraints and time windows from the MIP model implemented using CPLEX.

6.4.3 Our generator and dataset

The generator of Kummer improves in terms of being more realistic with respect to the original one by Mankowska et al., but it still presents some details that we can improve. Regarding the geographical part, it does not consider the real distribution of population in the city when sampling points, for which, albeit using real road distances, the distribution of the distances is still unrealistic, due to the fact that population in real contexts is not uniformly distributed in a geographical region.

Our proposal is to create a realistic instance generator that utilizes both the real distribution of population and actual road distances. It relies on the assumption that the density of patients in a given area is directly proportional to the density of the population. To make it fully realistic we should take into account additional factors such as age distribution and income level, which could potentially influence the demand for and the accessibility of homecare services. However, differently from population distribution, it is harder to retrieve such detailed data with a high level of resolution on a large-scale dataset, for which we consider the use of population density as a good trade-off between realism and convenience. Nonetheless, in principle, the model can be adapted to incorporate such factors if the relevant data is available.

The generator uses the GEOSTAT cartographic database provided by the Joint Research Centre and DG Regional Policy of Eurostat. The database partitions the entire area of the European Union and the adjacent countries into square cells measuring 1km^2. Each cell in the database contains information regarding the population residing within the cell. For our generator, we selected a variety of areas, including urban, rural, and mountainous regions, which possess varying features with respect to population, morphology, urban sprawl, and compactness. This diverse selection was made to create a wide range of instances. Within each chosen area, we sample N points, plus an additional point for the central office, specified by their latitude and longitude coordinates. The sampling process is weighted by the population of each cell, such that a point is more likely to be selected in a cell with higher population. Then, the obtained $N + 1$ points are matched to the nearest location accessible by road or street. This is necessary to calculate the road distances between pairs of points. We employ the Open Source Routing Machine (OSRM) to compute the time it takes to travel by car from one point u to another point v. The distances (u, v) and (v, u) are computed independently. The result is an asymmetric complete graph where the $N + 1$ vertices represent the N patients' homes

and the central office. The edges of the graph are weighted with actual road travel times as computed offline by the OSRM routing engine.

We generated a dataset composed of 200 instances sampling various Italian territories of different size and population density. The values of the main features, such as patients, caregivers, and services, are selected randomly for each instance (so that there are no groups).

From this dataset, we conducted a random selection of 30 instances for validation, while retaining the remaining 170 instances for training. The features of the validation instances are presented in Table 6.2. The time window for each patient spans between 60 and 180 minutes. Furthermore, in addition to the features outlined in Table 6.1, we also provide the distance radius from the central office, which was used during the generation of the instances.

6.4.4 File formats

Unfortunately, there is no single and well-established file format for the instances of this problem. Indeed, the dataset of Mankowska et al. uses two different formats, one for the instances of size up to 75 patients and one for the larger ones (from 100 patients upward). Kummer's dataset uses a third one, which is similar, but not identical to the second one by Mankowska et al.

All three formats are text-only and have a fixed-structure. They are easy to parse, but quite fragile and not very human readable. In addition, they contain some redundant data. For these reasons, we decided to move to a more robust and human-readable format based on JSON. Our JSON format is extensible to different, more complex versions of the problem, which is not the case for the previous ones. The solutions are also written in JSON, with their specific syntax.

In our repository located at `https://github.com/iolab-uniud/hhcrsp`, we provide a brief guide on the syntax of the input and output JSON files. This repository contains all the validation instances and our best solutions. Additionally, we offer a Python-based validator to ensure the correctness of instances and solutions.

Table 6.2: Features of new validation instances

Instance	P	S	C	Double service	Time window avg	Compatibility min–max	Distances avg	Radius [km]
0	299	8	42	6.7%	124.7	0.36–0.40	42.20	26
1	165	4	24	41.8%	117.7	0.42–0.58	23.61	19
2	229	8	33	47.6%	124.6	0.30–0.52	17.55	11
3	44	8	8	43.2%	106.4	0.25–0.50	22.58	19
4	258	4	30	17.8%	117.6	0.47–0.53	33.70	26
5	212	4	28	16.0%	120.0	0.39–0.61	39.42	29
6	213	8	30	28.2%	120.6	0.33–0.47	43.32	30
7	297	4	45	9.8%	116.4	0.49–0.51	39.53	29
8	247	8	36	32.4%	127.4	0.33–0.39	33.37	32
9	55	4	7	27.3%	115.9	0.43–0.57	21.55	15
10	76	4	10	30.3%	120.2	0.40–0.60	20.82	15
11	181	8	27	16.0%	125.1	0.22–0.52	34.48	33
12	130	4	21	24.6%	123.7	0.38–0.62	38.09	37
13	131	8	18	35.1%	124.7	0.22–0.44	23.95	15
14	213	4	33	42.3%	125.4	0.42–0.58	35.61	29
15	73	4	11	34.2%	114.7	0.36–0.64	21.50	15
16	145	4	15	13.8%	116.7	0.47–0.53	21.18	11
17	101	4	13	11.9%	118.1	0.38–0.62	24.96	26
18	356	4	51	42.7%	126.3	0.37–0.63	24.09	17
19	203	8	28	42.9%	126.1	0.29–0.43	24.94	18
20	78	4	12	46.2%	127.7	0.33–0.67	20.10	15
21	323	4	49	30.3%	123.7	0.35–0.65	30.86	26
22	255	4	37	25.9%	114.5	0.38–0.62	36.66	29
23	75	4	9	30.7%	123.8	0.44–0.56	18.34	15
24	270	4	33	11.9%	123.3	0.36–0.64	30.09	25
25	45	8	8	28.9%	113.7	0.13–0.63	22.82	18
26	191	4	26	18.9%	124.5	0.38–0.62	25.83	17
27	157	8	29	47.8%	125.7	0.17–0.55	16.74	10
28	378	8	51	19.0%	123.0	0.33–0.41	37.70	32
29	100	4	11	3.0%	115.5	0.45–0.55	32.87	21

Table 6.3: Parameter settings.

Name	Description	Value	Range
T_0	initial temperature	6.11	3–10
T_f	final temperature	0.13	0.05–0.15
α	cooling rate	0.9863	0.985-0.995
ρ	accepted moves ratio	0.063	0.05–0.15
σ_{SP}	probability of SP	0.04	0.0 – 0.3
σ_{IRS}	probability of IRS	0.04	0.0 – 0.3

6.5 Experimental analysis

Our solution method is implemented in C++ using the framework for local search *EasyLocal++* (v. 3)[2].

The experiments have been run on an Ubuntu Linux 22.4 machine with 4 cores Intel® i7-7700 (3.60 GHz), with a single core dedicated to each experiment.

6.5.1 Parameter tuning

The parameter tuning was performed using the tool JSON2RUN, similarly to Chapter 4.

The tuning procedure was performed in two stages. In Stage I, we tuned the parameters of SA, namely T_0, T_f, α, and ρ; in Stage II, we tuned the rates of the neighborhoods σ_*. In Stage I we used rates obtained from preliminary experiments, whereas in Stage II we used the SA parameters obtained in Stage I.

Tuning has been performed using only the training dataset proposed by Kummer (2021).

We conducted experiments with 30 Hammersley points and identified the best result, which is presented in Table 6.3. The initial parameters ranges were determined through preliminary experiments and are also provided in the table. During the tuning phase, the maximum number of iterations $\mathcal{I}$ for the simulated annealing (SA) algorithm was set to $5 \cdot 10^7$, leading to an average running time of approximately 88 seconds on our machine.

We notice that σ_{SP} and σ_{IRS} have a rather low value in the winning configuration, both equal to 0.04. Indeed, experiments using only neighborhood MovePatient obtain results not much inferior to the

[2]Available at `https://bitbucket.org/satt/easylocal-3`

multi-neighborhood setting. However, especially for the largest, and more challenging instances, the contribution of the two auxiliary neighborhood is statistically non-negligible. In fact, the configuration with $\sigma_{\mathsf{SP}} = 0$ and $\sigma_{\mathsf{IRS}} = 0$ was eliminated by the RACE procedure.

6.5.2 Comparative results on the dataset of Mankowska et al.

We excluded from our analysis group A as all 10 instances are always consistently solved to optimality, so they are not sufficiently challenging.

Regarding the parameter setting, due to the Metropolis acceptance criterion of SA, temperatures must be related to costs, which in turn are related to distances and tardiness. Given that in this dataset distances represent the biggest cost and they are consistently larger than the distances in the training instances by Kummer (2021), we rescaled the parameters T_0 and T_f. To this aim, we observed that the average distance in this dataset is 2.3 times bigger that in the training one, therefore we multiplied T_0 and T_f for 2.3 with respect to the values in Table 6.3, keeping all the others fixed as in the table.

The comparison of our results with the best-known results in the literature is presented in Tables 6.4 and 6.5 for different instance groups, namely B-D and E-G, respectively. Specifically, we compare our findings (3SA) with those obtained by Mankowska et al. (2014) and Lasfargeas et al. (2019) using Variable Neighborhood Search, as well as by Kummer et al. (2020, 2022) using Biased Random-Key Genetic Algorithms. Following usual conventions, the best (average) results are indicated in bold, while the optimal values (only found in group B) are underlined.

For each instance, we ran ten replicates of our solving procedure, with the number of iterations set at $\mathcal{I} = 10^8$. This corresponds to approximately 6 minutes of running time for the largest instances in group G. Although this allocated time is less than what others granted for the same instances, it led to significantly longer running times (up to 2 orders of magnitude) for us in the case of smaller instances. Nevertheless, our solver exhibits more linear time scalability in comparison to our competitors.

While a completely fair comparison is not feasible due to different running times and processors, it is evident that our best and average results outperform the others in nearly all cases. This observation is reinforced by the results averaged on each group, as indicated in the Avg rows in the tables. Furthermore, we consistently achieved optimal solutions for seven out of ten instances in group B. For the remaining three, we also find always the same

Table 6.4: Comparative results on instances of groups $B - D$.

Inst.	Kumm-22 LB	Mank-14 cost	t[s]	Lasf-19 best	avg	t[s]	Kumm-20 best	avg	t[s]	Kumm-22 best	avg	t[s]	3SA best	avg	t[s]
B_1	428.10	458.9	<1	434.1	552.8	53.1	428.10	428.26	8.6	428.10	428.53	0.8	428.10	**428.10**	73.9
B_2	476.05	476.2	<1	476.0	561.3	27.7	483.63	485.66	8.4	476.05	476.92	0.9	476.05	**476.05**	74.2
B_3	399.09	399.2	<1	399.1	527.6	63.5	402.80	402.80	9.0	402.80	409.29	1.0	399.09	**399.09**	75.6
B_4	411.30	576.0	<1	414.0	509.7	66.8	420.29	431.87	8.2	422.06	430.46	1.1	411.30	**411.30**	74.7
B_5	366.34	391.1	<1	385.6	496.9	13.7	372.16	374.24	8.2	369.44	375.15	1.0	366.34	**366.34**	74.2
B_6	405.58	534.7	<1	447.8	611.8	443.7	471.00	471.87	8.9	470.59	470.70	1.2	464.62	**464.62**	73.9
B_7	328.67	355.5	<1	328.7	398.8	61.5	328.67	**328.67**	9.5	328.67	**328.67**	0.9	328.67	**328.67**	72.7
B_8	357.68	357.8	<1	359.7	488.7	79.3	359.70	359.70	9.2	357.68	359.40	0.7	357.68	**357.68**	73.9
B_9	330.30	403.8	<1	404.1	483.4	62.1	402.67	404.27	10.0	404.11	404.29	0.9	402.67	**402.67**	75.6
B_{10}	420.99	500.4	<1	462.7	616.8	8.7	469.58	469.58	9.2	469.58	469.58	0.9	462.75	**462.75**	73.6
Avg		445.4		411.2	524.8	88.0	413.86	415.69	8.9	412.91	415.30	0.9	409.73	**409.73**	74.24
C_1	459.25	1123.6	<1	974.2	1350.4	96.2	965.15	975.59	36.9	969.11	**973.87**	3.1	943.73	1010.79	105.7
C_2	373.94	673.8	<1	605.1	685.5	106.4	583.39	590.48	39.0	584.18	587.00	2.9	569.12	**575.51**	103.6
C_3	390.48	642.4	<1	562.9	698.2	109.8	548.79	559.05	37.4	549.63	**552.52**	2.9	537.79	563.10	105.9
C_4	371.99	580.4	<1	521.9	630.4	112.4	519.91	530.59	36.2	520.13	524.15	3.0	495.17	**499.23**	104.1
C_5	464.97	754.6	<1	683.1	822.6	114.9	678.61	702.92	31.4	668.65	685.92	3.4	655.72	**666.50**	102.8
C_6	360.73	951.6	<1	854.6	1010.6	115.9	840.69	845.49	37.2	841.48	846.83	2.9	813.25	**831.62**	106.9
C_7	354.15	577.4	<1	529.2	572.5	109.4	534.85	540.39	42.2	533.92	541.88	3.4	511.89	**515.29**	103.0
C_8	375.52	540.6	<1	471.0	522.8	110.8	474.55	480.06	36.5	475.96	478.39	3.5	468.88	**470.26**	103.1
C_9	355.29	608.7	<1	551.1	642.7	115.4	534.30	551.14	42.8	545.18	558.54	3.4	527.69	**535.63**	105.1
C_{10}	431.18	679.3	<1	608.9	653.0	99.0	611.25	618.27	35.3	611.03	614.59	2.7	590.26	**590.27**	103.4
Avg		713.2		636.2	758.9	109.0	629.15	639.40	37.5	629.93	636.37	3.1	611.35	**625.82**	104.36
D_1	492.09	1321.8	5	1278.2	1498.8	143.0	1186.20	1209.62	93.7	1193.21	1215.79	8.0	1110.81	**1182.21**	136.4
D_2	384.68	892.7	4	746.9	914.3	168.7	693.28	718.03	82.9	679.58	695.99	7.2	653.73	**673.26**	134.7
D_3	380.05	819.4	4	678.6	817.8	155.4	635.67	651.35	102.2	644.16	650.22	8.5	613.12	**634.94**	135.4
D_4	418.94	877.4	4	809.7	1073.1	148.5	814.35	841.64	82.3	795.15	827.28	7.2	770.60	**784.07**	134.9
D_5	415.81	872.1	5	777.0	924.9	150.3	691.50	703.12	92.4	693.83	702.68	7.7	651.65	**661.18**	135.4
D_6	392.08	835.2	5	768.6	886.6	154.6	733.67	744.70	105.8	731.71	743.64	7.9	688.15	**692.65**	134.9
D_7	372.49	706.3	6	600.1	680.4	168.1	590.64	604.75	112.8	586.10	597.25	7.8	565.78	**580.54**	133.0
D_8	409.35	811.4	4	715.5	775.8	149.8	661.78	680.31	102.7	658.49	669.83	8.1	647.95	**662.06**	133.3
D_9	385.89	842.7	6	741.0	818.2	156.0	706.08	723.45	92.6	689.83	710.32	9.2	651.10	**660.49**	134.8
D_{10}	485.63	1306.6	3	1424.6	1867.7	173.1	1208.71	1290.56	77.7	1189.32	1280.92	6.9	1155.89	**1165.73**	136.3
Avg		928.6	4.6	854.0	1025.8	156.8	792.19	816.75	94.5	786.14	809.39	7.9	750.88	**769.71**	134.91

Table 6.5: Comparative results on instances of groups $E - G$.

	Kumm-22	Mank-14		Kumm-20			Kumm-22			3SA		
Inst.	LB	cost	t[s]	best	avg	t[s]	best	avg	t[s]	best	avg	t[s]
E_1	430.36	1604.9	17	1331.49	1352.32	193.6	1327.72	**1340.36**	17.2	1257.71	1357.71	166.1
E_2	444.88	1101.9	10	848.08	871.25	192.0	829.79	865.05	17.1	780.57	**791.59**	163.4
E_3	454.27	986.4	14	788.03	814.23	182.9	789.56	806.53	16.7	758.24	**775.46**	165.9
E_4	412.08	871.0	19	711.19	729.93	196.5	723.87	728.96	14.8	679.57	**696.71**	166.8
E_5	416.62	1018.0	19	781.50	803.86	182.3	780.04	817.13	17.0	714.00	**739.40**	164.9
E_6	416.60	1003.0	19	790.47	804.16	177.6	779.82	793.68	18.3	751.26	**762.77**	167.0
E_7	389.57	921.1	20	711.11	733.91	191.8	705.79	715.46	18.0	680.55	**692.67**	165.4
E_8	433.89	884.6	19	752.35	761.97	168.8	733.90	750.59	17.2	708.08	**717.51**	167.3
E_9	446.49	1131.7	18	921.78	951.91	163.1	893.35	916.56	16.4	840.14	**869.21**	166.2
E_{10}	455.07	1053.6	11	825.24	845.10	174.5	822.85	841.57	16.0	782.76	**802.24**	163.9
Avg		1057.6	16.6	846.12	866.86	182.3	838.67	857.59	16.9	795.29	**820.53**	165.69
F_1	548.88	1721.4	889	1401.96	1425.97	745.5	1311.10	1351.20	124.4	1229.68	**1274.23**	298.8
F_2	543.32	1763.8	909	1336.33	1383.58	812.1	1298.31	1337.41	121.7	1214.39	**1248.10**	296.8
F_3	547.64	1549.6	868	1263.39	1285.81	780.3	1215.96	1272.23	116.5	1136.95	**1170.20**	298.0
F_4	531.84	1420.4	1321	1124.24	1146.22	901.9	1100.66	1134.66	136.0	1027.68	**1070.69**	297.4
F_5	538.14	1701.9	1145	1329.29	1365.17	826.1	1298.55	1331.09	119.8	1214.61	**1240.69**	298.7
F_6	518.47	1639.7	836	1332.14	1373.32	649.8	1292.52	1368.41	109.6	1231.47	**1264.24**	300.6
F_7	512.98	1384.3	1294	1131.27	1157.35	817.0	1084.57	1125.37	120.5	1063.33	**1093.83**	296.4
F_8	536.15	1544.6	924	1132.77	1165.15	716.4	1123.22	1140.42	107.7	1077.12	**1109.37**	298.5
F_9	543.16	1572.9	1642	1311.43	1345.01	770.4	1263.19	1344.62	125.4	1183.69	**1215.89**	299.3
F_{10}	546.84	1581.0	1326	1418.53	1446.35	740.3	1383.08	1419.76	119.6	1274.71	**1306.27**	301.0
Avg		1588.0	1115.4	1278.14	1309.39	776.0	1237.12	1282.52	120.1	1165.36	**1199.35**	298.55
G_1	612.37	2248.0	7200	1778.54	1855.17	1949.8	1744.14	1824.34	439.4	1668.14	**1707.70**	456.7
G_2	605.84	2316.1	7200	1824.74	1897.99	2115.1	1709.70	1799.78	519.5	1633.70	**1668.64**	454.0
G_3	614.20	1885.3	7147	1514.23	1546.53	1935.1	1464.69	1511.86	461.6	1387.44	**1438.69**	453.8
G_4	604.30	2023.2	7200	1564.42	1599.39	2137.6	1508.94	1569.01	529.0	1448.65	**1491.84**	455.3
G_5	633.66	2247.6	7200	1698.28	1749.10	1840.9	1652.88	1681.01	466.6	1547.88	**1585.38**	458.3
G_6	621.46	2144.4	7200	1714.38	1777.39	2014.4	1681.64	1719.18	570.5	1630.98	**1680.26**	457.6
G_7	602.42	1971.5	6934	1640.07	1677.92	1844.3	1536.00	1604.96	522.4	1494.00	**1534.86**	453.2
G_8	618.74	1987.4	7200	1547.63	1583.86	1799.1	1498.38	1535.90	531.7	1444.88	**1478.84**	453.3
G_9	662.70	2415.5	7023	1942.21	1972.48	1810.7	1850.07	1976.27	446.6	1762.51	**1799.20**	457.9
G_{10}	633.76	2373.4	7003	1872.08	1932.27	1649.6	1785.37	1868.56	482.8	1664.70	**1714.07**	456.9
Avg		2161.2	7130.7	1709.66	1759.21	1909.7	1643.18	1709.09	497.0	1568.29	**1609.95**	455.7

value, but it is not known whether it is optimal.[3]

6.5.3 Comparative results on Kummer's dataset

Tables 6.6 to 6.8 present the comparative results for the second dataset, with group I-10 removed. As explained above, the validation dataset is composed by the 10 hardest instances, and 10 random ones out of 2200 generated for each group. In our tables, the hardest instances are on the left, the random ones on the right. The comparison here is only against the results of Kummer (2021) as these instances have been proposed recently and are not considered by other studies.

We can see that our results outperform the previous ones on most instances, in particular on the largest ones. The only group in which the previous results are cumulatively superior is the smallest one (25 patients) of hard instances.

As for the previous dataset, our running times, corresponding to 10^8 iterations, are longer than Kummer (2021) for small instances and shorter for the large ones.

6.5.4 Results on our new dataset

We now show the results obtained for the validation instances of our new dataset. In this case, we have no comparative results available, so we only show our values for future comparisons. We report in Table 6.9 the average for 10 runs of the cost (obtained using the three weights equal to 1/3) and the individual cost components (distance, total tardiness, highest tardiness). In order to give a more qualitative view of the costs, we also show the average distance per caregiver and the average tardiness per provided service. The last row shows the averages upon all instances for these measures.

We see that the average distance (in minutes) for caregiver is 122.8, meaning that an operator spends about two hours of their working day traveling between patients and to the central office. Average tardiness is 0.3 minutes, which is perfectly acceptable, although this value also considers all the services with no tardiness. The average highest tardiness is 11.8 minutes, which is quite low and tolerable in general. We also see that there are no instances with zero tardiness, showing that this objective is indeed binding and significant.

[3] All our results have been validated using the MIP model by Mankowska et al. (2014), implemented in CPLEX and kindly supplied to us by Alberto Kummer.

Table 6.6: Comparative results on instances of groups $25 - 50$.

Inst.	Kumm-21 best	Kumm-21 avg	Kumm-21 t[s]	3SA best	3SA avg	3SA t[s]	Inst.	Kumm-21 best	Kumm-21 avg	Kumm-21 t[s]	3SA best	3SA avg	3SA t[s]
25_5_22_1.0_C_C	820.33	**831.47**	49.07	820.33	862.68	69.33	25_5_80_0.8_C_RC	436.67	**436.67**	49.07	436.33	443.13	71.13
25_5_69_1.6_R_RC	1149.00	**1149.00**	49.01	1146.00	1162.92	68.05	25_5_48_1.0_C_C	275.00	277.18	49.53	272.0	**272.19**	71.66
25_5_47_0.8_R_C	848.00	**848.00**	49.15	842.33	850.07	71.61	25_5_42_0.8_C_RC	287.67	**287.67**	49.41	287.33	288.70	72.88
25_5_37_0.8_R_RC	1434.67	**1434.67**	49.23	1434.67	1442.24	71.04	25_5_56_0.8_R_RC	178.33	**178.80**	50.09	177.33	179.29	72.33
25_5_37_1.0_R_RC	1434.67	**1434.67**	49.15	1434.67	1442.47	70.98	25_5_76_1.2_C_C	183.67	**183.67**	49.65	183.67	185.27	71.44
25_5_26_0.8_C_C	956.33	**956.33**	49.20	952.33	996.70	71.72	25_5_70_1.0_R_C	146.00	146.00	50.02	141.67	**142.23**	70.67
25_5_6_0.8_R_C	681.00	687.60	49.55	678.67	**682.94**	72.03	25_5_21_1.6_R_RC	204.67	204.67	50.04	197.67	**200.20**	68.81
25_5_69_1.6_R_C	695.00	**696.06**	49.06	695.00	724.22	69.57	25_5_73_0.4_R_RC	151.00	152.85	49.59	150.00	**150.00**	70.92
25_5_42_0.8_R_RC	1161.67	1168.45	48.98	1158.33	**1165.91**	69.98	25_5_97_1.0_R_R	258.67	**258.67**	49.41	258.67	**258.67**	71.36
25_5_96_0.8_R_C	707.67	**707.67**	49.49	697.67	718.98	72.68	25_5_49_0.4_C_C	138.67	138.69	49.86	137.67	**137.67**	71.79
Avg.	988.83	**991.39**	49.19	986.00	1004.91	70.70	Avg.	226.04	226.49	49.66	224.23	**225.74**	71.30
50_10_80_1.0_C_C	826.67	829.05	58.27	815.67	**818.54**	102.56	50_10_3_1.6_C_C	262.00	266.25	61.58	251.0	**252.83**	99.30
50_10_81_1.0_R_RC	999.67	1010.04	58.12	986.33	**994.50**	100.58	50_10_53_1.2_R_C	250.67	254.92	60.59	247.0	**251.96**	101.15
50_10_80_1.0_C_RC	1057.33	1063.25	58.28	1045.0	**1048.74**	103.16	50_10_62_1.2_R_C	211.33	218.25	62.81	204.0	**205.80**	100.40
50_10_97_0.8_C_RC	759.67	**770.72**	59.21	749.67	779.74	102.47	50_10_74_1.0_R_RC	290.33	**291.93**	60.31	289.67	297.64	101.22
50_10_32_1.2_R_C	442.33	463.30	60.76	431.33	**454.56**	101.79	50_10_61_1.0_R_RC	276.33	279.62	62.71	270.0	**271.57**	100.68
50_10_3_0.8_R_C	589.00	596.18	60.90	585.67	**589.31**	101.19	50_10_88_1.6_C_RC	317.67	320.93	61.92	308.0	**312.92**	100.25
50_10_80_0.4_C_C	762.67	762.84	58.77	752.33	**757.70**	101.40	50_10_60_0.4_C_RC	266.33	267.38	62.30	261.0	**265.28**	99.99
50_10_26_1.0_C_RC	610.33	**611.10**	59.83	607.0	612.78	102.94	50_10_2_1.0_C_R	345.00	348.70	60.77	342.33	**346.11**	100.90
50_10_26_1.2_R_C	312.00	317.02	60.91	303.0	**310.86**	100.46	50_10_100_1.6_C_C	159.00	159.00	61.92	154.67	**156.13**	99.73
50_10_44_1.6_R_C	279.33	284.55	62.56	270.0	**279.44**	99.97	50_10_33_0.4_R_RC	279.67	280.64	62.12	274.33	**280.54**	100.56
Avg.	663.90	670.80	59.76	654.60	**664.62**	101.65	Avg.	265.83	268.76	61.70	260.20	**264.08**	100.42

Table 6.7: Comparative results on instances of groups $75 - 100$.

Inst.	Kumm-21 best	Kumm-21 avg	Kumm-21 t[s]	3SA best	3SA avg	3SA t[s]	Inst.	Kumm-21 best	Kumm-21 avg	Kumm-21 t[s]	3SA best	3SA avg	3SA t[s]
75_15_97_1.0_R_RC	798.00	814.52	72.50	779.67	**806.37**	131.48	75_15_97_0.4_C_C	480.00	483.37	74.83	462.0	**470.77**	131.27
75_15_35_0.8_R_C	719.00	730.60	72.90	704.0	**717.74**	130.02	75_15_63_1.0_R_RC	408.00	413.10	78.22	395.0	**400.63**	128.86
75_15_35_1.0_C_RC	1051.33	1058.83	73.27	1036.0	**1046.3**	131.12	75_15_5_0.8_C_RC	439.67	446.17	78.09	426.67	**439.03**	129.83
75_15_97_1.0_C_C	612.33	635.02	74.86	593.0	**604.67**	133.54	75_15_19_0.8_R_C	368.67	377.88	77.86	361.67	**373.88**	129.25
75_15_79_1.2_R_C	404.33	421.37	79.09	394.33	**400.69**	126.11	75_15_84_0.8_R_RC	392.33	401.47	76.96	381.67	**389.83**	128.59
75_15_10_1.6_R_C	376.00	390.67	74.60	343.0	**357.67**	128.95	75_15_30_1.0_R_C	303.67	309.13	80.97	292.0	**297.77**	128.82
75_15_88_1.6_R_C	461.33	480.05	79.34	449.67	**456.69**	129.86	75_15_32_1.2_R_RC	353.67	357.68	76.31	338.67	**344.21**	130.36
75_15_98_1.6_R_C	375.67	393.70	76.71	359.33	**373.92**	129.69	75_15_40_0.8_R_RC	423.67	429.10	82.88	409.33	**418.30**	129.89
75_15_21_1.6_R_C	384.67	389.02	81.12	363.67	**379.66**	126.79	75_15_63_0.4_R_C	317.33	322.50	78.69	303.67	**311.06**	130.30
75_15_87_1.6_C_C	386.33	388.57	76.80	358.67	**372.51**	129.78	75_15_65_1.0_C_C	257.00	259.53	78.81	252.0	**256.60**	128.89
Avg.	556.90	570.23	76.12	538.13	**551.62**	129.73	Avg.	374.40	379.99	78.36	362.27	**370.21**	129.61
100_20_8_1.2_R_C	607.33	**635.73**	105.73	606.67	636.84	155.84	100_20_68_1.6_R_C	304.67	313.48	115.38	306.00	**310.07**	155.21
100_20_63_1.2_R_C	548.33	**570.03**	107.07	552.00	577.51	152.74	100_20_49_0.4_C_C	525.67	536.42	103.04	512.33	**526.34**	157.46
100_20_39_1.6_R_C	554.33	567.50	99.28	540.67	**554.50**	154.97	100_20_61_0.4_R_RC	471.33	477.83	110.32	450.00	**460.97**	157.76
100_20_76_1.6_R_C	500.00	510.58	98.36	476.00	**490.37**	153.07	100_20_88_1.2_C_C	269.67	273.62	99.41	263.33	**268.83**	158.00
100_20_57_1.6_R_C	505.67	523.40	105.03	488.00	**500.22**	156.40	100_20_9_1.0_R_R	554.67	564.48	110.90	545.00	**557.51**	158.10
100_20_25_0.8_R_C	654.67	670.80	97.75	621.33	**641.31**	155.36	100_20_35_1.6_C_RC	442.67	453.77	103.82	422.00	**434.11**	157.97
100_20_77_1.0_R_C	613.67	640.15	105.58	606.67	**623.92**	152.95	100_20_10_1.2_R_C	300.00	308.20	103.62	297.67	**302.99**	158.87
100_20_11_1.2_R_C	535.33	551.07	95.21	504.67	**522.10**	156.48	100_20_65_0.4_R_RC	495.00	503.87	104.77	481.33	**491.31**	159.08
100_20_30_1.0_R_C	621.67	635.85	101.69	604.33	**629.99**	153.57	100_20_99_0.4_C_RC	487.00	492.63	107.93	469.33	**476.18**	155.73
100_20_23_1.2_R_C	454.00	467.57	107.64	438.33	**446.83**	153.67	100_20_54_1.2_R_RC	398.67	410.53	101.42	399.67	**408.83**	157.94
Avg.	559.50	577.27	102.33	543.87	**562.36**	154.51	Avg.	424.94	433.48	106.06	414.67	**423.71**	157.61

Table 6.8: Comparative results on instances of groups 200 – 400.

Inst.	Kumm-21 best	Kumm-21 avg	Kumm-21 t[s]	3SA best	3SA avg	3SA t[s]	Inst.	Kumm-21 best	Kumm-21 avg	Kumm-21 t[s]	3SA best	3SA avg	3SA t[s]
200_40_52_1.6_R_C	932.00	972.72	276.99	879.67	**908.91**	289.15	200_40_17_1.6_C_C	569.67	577.58	343.10	545.0	**559.03**	289.65
200_40_7_1.2_R_C	1081.33	1113.22	263.19	957.0	**1005.39**	289.61	200_40_75_1.6_R_C	679.00	693.62	296.38	648.33	**662.33**	293.56
200_40_30_1.6_R_C	923.67	953.12	280.84	878.67	**917.73**	287.55	200_40_80_1.0_C_C	635.33	646.52	372.74	623.67	**639.77**	293.87
200_40_76_1.2_R_C	1133.33	1194.73	301.42	1081.0	**1111.1**	288.66	200_40_82_1.0_C_R	944.00	977.73	296.24	901.33	**925.96**	292.77
200_40_7_1.6_R_C	847.67	876.15	263.28	796.0	**814.70**	293.98	200_40_69_1.0_C_R	991.67	1015.40	341.85	949.0	**961.77**	295.80
200_40_51_1.0_R_C	1136.67	1206.20	292.73	1072.67	**1097.72**	287.91	200_40_60_0.4_C_RC	870.67	900.08	268.88	835.0	**853.31**	292.97
200_40_17_1.6_R_C	835.00	858.85	340.20	791.67	**810.12**	286.58	200_40_10_1.6_C_RC	843.33	859.15	323.51	809.0	**827.16**	291.85
200_40_40_1.0_R_C	913.00	942.42	297.18	863.0	**886.16**	288.03	200_40_98_1.6_C_RC	830.33	844.53	311.00	797.33	**812.43**	294.15
200_40_71_1.6_R_C	857.67	883.70	274.77	811.33	**828.67**	290.08	200_40_100_0.4_C_C	711.67	726.58	342.13	688.67	**700.76**	293.94
200_40_98_1.2_R_C	936.00	973.03	314.45	926.0	**944.30**	287.59	200_40_71_1.6_C_C	458.67	468.78	298.01	437.0	**450.84**	297.45
	959.63	997.41	290.50	905.70	**932.48**	288.91		753.43	771.00	319.38	723.43	**739.34**	293.60
300_60_4_1.6_R_C	1568.67	1629.28	881.04	1479.33	**1515.52**	440.05	300_60_7_1.2_R_RC	1434.33	1486.10	627.71	1353.67	**1386.63**	451.88
300_60_89_1.6_R_C	1431.33	1507.85	672.63	1382.67	**1414.86**	442.91	300_60_75_1.2_C_RC	1088.33	1134.57	762.75	1057.33	**1080.89**	459.97
300_60_82_1.2_R_C	1499.67	1586.38	729.01	1428.67	**1457.96**	443.74	300_60_26_0.4_C_RC	1284.67	1343.53	689.58	1248.67	**1268.22**	455.09
300_60_95_1.2_R_C	1596.67	1696.10	845.22	1506.33	**1555.93**	443.89	300_60_68_0.8_C_RC	1182.67	1230.22	847.66	1174.33	**1193.30**	457.47
300_60_22_1.2_R_C	1458.67	1522.58	823.69	1430.0	**1461.66**	443.82	300_60_4_1.0_R_C	1016.00	1078.32	890.83	966.67	**990.40**	454.24
300_60_85_1.2_R_C	1412.33	1479.60	715.23	1307.67	**1345.38**	447.29	300_60_73_0.8_R_RC	1175.67	1234.60	740.73	1082.67	**1117.81**	456.54
300_60_81_1.6_R_C	1230.67	1313.85	785.84	1185.0	**1204.56**	447.06	300_60_45_1.6_C_C	726.00	747.50	834.72	688.67	**698.48**	457.39
300_60_23_1.2_R_C	1271.33	1322.22	823.18	1218.0	**1251.61**	451.73	300_60_46_0.4_R_C	1068.00	1113.27	801.14	1041.67	**1060.11**	460.30
300_60_34_1.2_R_C	1635.00	1709.00	801.04	1451.0	**1481.31**	445.53	300_60_81_0.8_C_C	814.00	835.68	806.91	778.33	**796.78**	461.93
300_60_69_1.0_R_C	1530.00	1579.32	789.10	1410.0	**1439.10**	444.43	300_60_48_0.8_C_C	826.00	850.58	811.38	812.33	**825.24**	464.62
	1463.43	1534.62	786.60	1379.87	**1412.79**	445.05		1061.57	1105.44	781.34	1020.43	**1041.79**	457.94
400_80_59_1.0_R_C	2400.67	2464.35	1602.10	2135.67	**2174.41**	623.95	400_80_54_1.2_C_RC	1562.33	1675.90	1757.88	1522.67	**1542.97**	629.73
400_80_16_1.2_R_C	2201.33	2311.37	1631.95	1962.33	**1995.44**	614.56	400_80_20_0.4_C_RC	1636.33	1731.70	1430.20	1581.67	**1612.38**	641.43
400_80_51_1.0_R_C	2237.33	2425.50	1522.17	2074.67	**2126.64**	612.96	400_80_61_0.4_C_RC	1625.00	1666.07	1627.19	1556.33	**1584.70**	633.24
400_80_97_1.6_R_C	1759.67	1820.27	1733.84	1561.0	**1607.43**	618.26	400_80_33_1.0_C_RC	1527.33	1620.13	1718.09	1478.33	**1501.32**	641.89
400_80_30_1.6_R_C	1834.00	1908.13	1474.27	1678.0	**1700.80**	612.68	400_80_95_1.0_C_R	1547.33	1600.53	1660.34	1470.0	**1504.36**	638.15
400_80_34_1.0_R_C	2064.67	2145.38	1605.12	1835.33	**1880.71**	612.48	400_80_65_1.0_C_R	1618.00	1689.10	1650.58	1504.0	**1531.89**	637.37
400_80_92_1.6_R_C	1791.33	1851.02	1495.99	1599.33	**1628.06**	625.25	400_80_45_0.4_C_C	1265.67	1324.67	1526.54	1186.0	**1216.59**	638.65
400_80_91_1.2_R_C	1909.33	2025.52	1608.22	1826.33	**1864.77**	616.77	400_80_22_1.6_C_C	948.00	1001.80	1725.22	869.0	**887.37**	643.22
400_80_28_1.2_R_C	2079.00	2172.27	1553.66	1913.67	**1943.40**	621.14	400_80_77_1.2_R_RC	1447.33	1504.32	1904.06	1298.33	**1330.06**	638.64
400_80_81_1.6_R_C	1743.00	1815.32	1710.40	1600.0	**1624.63**	612.49	400_80_60_0.8_C_C	1065.67	1128.53	1413.34	1030.33	**1051.96**	645.96
	2002.03	2093.91	1593.77	1818.63	**1854.63**	617.05		1424.30	1494.28	1641.34	1349.67	**1376.36**	638.83

Table 6.9: Results on the new instances

Instance	Cost	Distance	Distance per caregiver	Total tardiness	Tardiness per service	Highest tardiness
0	2058.0	6139.7	146.2	24.2	0.1	10.0
1	864.9	2583.5	107.6	7.6	0.0	3.6
2	915.6	2741.5	83.1	3.8	0.0	1.6
3	367.9	1101.8	137.7	1.1	0.0	0.8
4	1317.1	3934.5	131.2	13.4	0.0	3.3
5	1251.4	3746.1	133.8	5.6	0.0	2.4
6	1987.1	5892.9	196.4	55.4	0.2	13.0
7	1598.8	4695.8	104.4	52.5	0.2	48.0
8	1654.5	4946.2	137.4	12.4	0.0	4.8
9	301.7	902.3	128.9	1.8	0.0	1.1
10	428.4	1253.8	125.4	22.2	0.2	9.2
11	1219.7	3626.0	134.3	24.5	0.1	8.5
12	925.6	2765.3	131.7	7.6	0.0	4.0
13	693.5	2065.1	114.7	11.1	0.1	4.4
14	1327.6	3965.5	120.2	11.8	0.0	5.5
15	471.4	1274.8	115.9	106.7	1.1	32.7
16	514.4	1540.4	102.7	1.9	0.0	1.0
17	532.9	1589.3	122.3	5.6	0.0	3.7
18	1511.6	4525.5	88.7	6.4	0.0	2.8
19	1275.0	3815.9	136.3	6.6	0.0	2.5
20	447.6	1330.9	110.9	7.7	0.1	4.2
21	1897.4	5670.4	115.7	17.4	0.0	4.3
22	1644.4	4886.6	132.1	35.7	0.1	10.9
23	338.1	1009.6	112.2	3.2	0.0	1.4
24	1309.6	3910.4	118.5	13.2	0.0	5.3
25	489.5	917.9	114.7	403.7	7.0	147.0
26	1024.5	3059.0	117.7	10.7	0.0	3.8
27	620.0	1843.7	63.6	11.6	0.0	4.8
28	2249.5	6722.0	131.8	21.6	0.0	5.0
29	624.1	1847.9	168.0	18.7	0.2	5.7
avg	1062.1	3143.5	122.8	30.9	0.3	11.8

Table 6.10: Comparison on new instances with different combinations of the weights

	(1,1,1)	(1,10,1)		(1,1,10)		(1,10,10)	
	value	value	gap	value	gap	value	gap
Distance	3143.5	3263.9	3.8%	3189.2	1.5%	3263.6	3.8%
Distance per caregiver	122.8	127.3	3.7%	124.6	1.5%	127.6	3.9%
Total tardiness	30.9	25.7	-16.8%	29.3	-5.0%	21.5	-30.3%
Tardiness per service	0.33	0.32	-3.3%	0.40	20.9%	0.30	-8.8%
Highest tardiness	11.8	11.2	-5.2%	6.7	-43.1%	8.5	-28.4%

As an example, a solution for instance 3 is shown in Fig. 6.4. We see that there are only two late patients (p_1 and p_{36}), highlighted in white diagonal stripes. On the contrary, there are many early arrivals (e.g., p_8 and p_{14}) that cause idleness of the caregiver, which however are not penalized in the objective function.

Finally, we discuss the trade-off between travel time and tardiness. To this aim, we run some experiments using alternative weights. In particular, we keep the distance weight fixed and we multiply the other two by 1 and 10 alternatively.

In Table 6.10 we show the average results in comparison with the ones with the original weights. In particular, the column (1,1,1) represents the original weights, the column (1,10,1) the ones where the total tardiness is weighted ten times and the highest tardiness has the original weight, and so on. Looking at columns (1,10,1) and (1,1,10), we see that, unsurprisingly, the objective component with the weight increased improves its score at the expenses of the distance traveled. The other tardiness-related component is also improved, as the two of them are connected. Notice that the tardiness per service for weights (1,1,10) actually increases by 20.9%, but this is the average of small quantities that is sensitive to a few large values. For weights (1,10,10) all tardiness indicators decrease as expected.

However, the differences are quite small in absolute terms, showing that there is no strong trade-off between the components. This means that, due to patients' time windows, we cannot completely eliminate the tardiness by just increasing the traveling time of the caregivers, but we should rather increase their number.

Figure 6.4: Solution of instance 3

6.6 Conclusions

In this chapter, we have introduced a Multi-Neighborhood approach that utilizes a larger neighborhood set compared to previous methods proposed in the literature. Despite the larger size of the compound neighborhood, the efficiency of our method is maintained through the random selection criterion inherent in the SA algorithm, which avoids exhaustive exploration of the entire neighborhood.

Remarkably, our MNSA approach has yielded favorable comparisons against all state-of-the-art methods in the literature, achieving the best-known results for the majority of instances. While small instances may experience slightly longer running times, these remain within acceptable limits.

An additional contribution of this work is the introduction of a novel benchmark set that extends the existing literature in terms of instance challenges and feature diversity. This new benchmark set not only serves to evaluate the performance of our proposed approach but also provides a more comprehensive and extensive platform for future studies in the field.

Chapter 7

Capacitated Dispersion Problem

In this chapter, we consider the Capacitated Dispersion Problem (CDP), which is a classic formulation within the family of *diversity* problems on graphs that underlies many real-world problems, such as facility location and network analysis.

The problem consists in finding a set of nodes in an undirected complete weighted graph that maximizes the minimum distance among selected nodes, subject to a capacity constraint on the selection.

This problem has been tackled in several recent works (Peiró et al., 2021; Martí et al., 2021; Mladenović et al., 2022; Lu et al., 2023), mainly using metaheuristic approaches. The problem comes along with a public testbed, collected by Peiró et al. (2021), that has been used as benchmark in all mentioned papers.

We propose a local search approach that uses a portfolio of neighborhoods and a Simulated Annealing metaheuristic to guide the search.

The neighborhoods that we use are gathered from the literature, though suitably modified and adapted for our context. Our approach is enriched by a lexicographic objective function that facilitates the navigation on the plateaus.

The experimental analysis shows that our multi-neighborhood method, properly engineered and tuned, is able to outperform the aforementioned previous techniques on the available benchmark.

In addition, we highlight severe limitations on the benchmark itself and we propose a novel, more diverse and challenging dataset that aims to overcome these limitations, and could become an additional benchmark for

this problem for the future.

For this new dataset, properly split into test and validation instances, we provide the results of both our method and of the method by Lu et al. (2023) obtained by compiling and running their code, which is publicly available. The results confirm that our method is able to obtain better results on this dataset as well.

Finally, we propose a new MILP model and its implementation using CPLEX. This model, inspired by the work by Erkut and Neuman (1991) on similar problems, uses less variables and less constraints than the previous ones proposed by Peiró et al. (2021) and Martí et al. (2021), and it has been able to provide good bounds and optimal results for small and medium size instances. In addition, using very long runs (12 hours), it also obtains good results on large instances.

The new dataset and all our best solutions are available at `https://github.com/iolab-uniud/cdp` for inspection and future comparisons.

7.1 Related work

Diversity and dispersion problems have been studied intensively starting from the 70s (see, e.g., Shier, 1977). We refer to the recent survey by Martí et al. (2022) for an overview of the various formulations and the search methods used to solve them.

The specific version considered in this chapter, called CDP, has been introduced by Rosenkrantz et al. (2000), who proposed a greedy algorithm and an approximation scheme for both CDP and other versions of the problem. In particular, they prove an approximation factor of 2 for the case in which the distances satisfy the triangular inequality. This result is of theoretical interest, but with limited practical use.

The CDP has been recently addressed by using metaheuristic approaches. Peiró et al. (2021) propose a technique based on GRASP, Variable Neighborhood Search, and Strategic Oscillation, and also a mathematical model. They adapted the datasets from other problems, and then they compared their method with the CPLEX implementation of the mathematical model, and with the greedy algorithm of Rosenkrantz et al. (2000).

Subsequently, Martí et al. (2021) proposed a metaheuristic method based on scatter search and also developed a new mathematical model. They compared their search method and their model with the ones by Peiró et al. (2021).

Lu et al. (2023) apply a solution-based Tabu Search using three distinct neighborhoods: insert, remove and swap. In order to speed up the check of equality of the current solution with tabu ones, they employ hash functions that identify eligible candidate solutions.

The current state-of-the-art results for the CDP are provided by Mladenović et al. (2022), that developed various versions of Variable Neighborhood Search, namely Basic VNS, General VNS and General Skewed VNS. In addition to the neighborhoods used by Lu et al. (2023), they also use the neighborhoods 2-out-1-in and 1-out-2-in, that insert or remove two nodes at a time.

7.2 Problem definition

We are given an undirected graph $G = (V, E)$, where V is a set of nodes, and E is a set of edges between nodes in V. Each node i is assigned a capacity c_i and each edge is assigned a value d_{ij} representing the *distance* between nodes i and j. We are also given a single value B representing the total minimum capacity requested for the solution. All values are reals.

The graph is assumed complete and all distances are non-negative, but they are not assumed Euclidean, and they do not need to satisfy the triangular inequality.

The problem consists in selecting a set of nodes $S \subseteq V$ such that the sum of the capacities of the nodes in S is at least equal to B and the minimum distance between nodes in S is maximized.

Our mathematical model is based on binary variables $x_i \in \{0, 1\}$ with $i \in V$ that take value 1 if the node i is selected in the solution, 0 otherwise. In order to express the objective to maximize the minimum distance between the selected nodes in the linear model, we introduce a new, real-valued variable $\bar{d} \in \mathbb{R}$ that represents the minimum distance between the selected nodes. The objective function is then $\max \bar{d}$ and the model is the following one.

$$\max \quad \bar{d} \tag{7.1}$$

$$\sum_{i \in V} c_i x_i \geq B \tag{7.2}$$

$$\bar{d} \leq d_{ij} + M(1 - x_i) + M(1 - x_j) \qquad \forall i, j, i < j \in V, i \neq j \tag{7.3}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V \tag{7.4}$$

$$\bar{d} \geq 0 \tag{7.5}$$

The first constraint in Eq. (7.2) simply imposes that the sum of the capacities c_i of the selected nodes is at least the minimum capacity B. The second set of constraints in Eq. (7.3) imposes that, for every pair of nodes (i, j), the minimum distance $\bar{d}$ is at most equal to the distance d_{ij}, but only if both nodes are selected.

To express this condition, we use the BigM technique, with the constant M that takes a suitably high value, which is multiplied by $(1 - x_i)$ and $(1 - x_j)$. In this expression, if one of the two nodes is not selected, that is, either $(1 - x_i)$ or $(1 - x_j)$ is 1, then the constraint says that $\bar{d}$ is free to take any value below $M + d_{ij}$, or below $2M + d_{ij}$ if both nodes are unselected. If, on the other hand, x_i and x_j are selected, both terms that multiply M take value 0 and $\bar{d}$ is constrained by the distance between the two nodes d_{ij}. Given that the constraint is repeated for all pairs of nodes (i, j), the distance $\bar{d}$ is constrained by the minimum distance between the selected nodes.

This MILP model is inspired by the ones proposed by Erkut and Neuman (1991), suitably adapted to CDP. It is significantly more compact than the one by Peiró et al. (2021) that uses an additional variable y_{ij} for each pair (i, j), that takes value one if both x_i and x_j have value one. It also uses less variables and less constraints than the one by Martí et al. (2021), which however is based on a completely different paradigm.

7.3 Solution method

We propose a Multi-Neighborhood local search approach for CDP. In the subsequent sections, we will outline the fundamental components of the local search paradigm, which include the *search space*, the *initial solution strategy*, the *neighborhood relations*, the *cost function* and the *metaheuristic* that drives the search process.

7.3.1 Search space

Our search space is characterized by two disjoint sets. The first is S, that coincides with the solution, that we also name *selection list*, and that comprehends the nodes that are currently selected. We also store the complementary set $C = V \setminus S$, named *candidate list* because it is the set of nodes that are candidates for insertion in the current solution.

We assume that $|S| \geq 2$, that is, a solution always contains at least two nodes and one edge. While in theory it might exist a solution with only one node that alone satisfies the capacity requirement, in the current datasets this never happens, therefore we can safely prevent the case of having a

solution with one single node, which would have an undefined value of the objective function.

With regard to the capacity constraint, we do not impose its satisfaction and thus we let the search explore both the feasible and infeasible region.

To speed up the computation and evaluation of moves, we define several redundant data structures that complement the two sets S and C. For every node, regardless whether it is or not in the selection list, we store the value of the minimum distance edge that connects the node to the selection list into a variable $d_S(v)$ and we compute it as $d_S(v) = \min_{u \in S} d_{uv}$. We also keep the counter $|d_S(v)|$, telling how many edges, for every node, with such distance there are. Finally, recalling that the objective function of the problem is defined as $f(S) := \min_{u,v \in S} d_{uv}$, we define a set $E_R \in E$ of the edges that are determining the value of the objective function, that is, the set of edges between nodes in S with distance exactly equal to $f(S)$. We call these edges as the *bottleneck* edges. Rather than keeping track of the full set E_R, we only store and update the value $|E_R|$.

7.3.2 Initial solution strategy

The initial solution is generated starting from S equal to the empty set. Then, until the capacity requirement B is satisfied, and in any case while $|S| < 2$, we iteratively add random nodes $v \in C$ and we insert them in S. Therefore, the initial solution is generated completely at random, just ensuring that the capacity requirement is satisfied.

7.3.3 Multi-neighborhood

We propose a Multi-Neighborhood approach that makes use of three neighborhoods:

- Insert: the move $\mathsf{Insert}\langle v \rangle$ inserts a node v from the candidate list into the selection list S.
  Preconditions: $v \in C$.

- Remove: the move $\mathsf{Remove}\langle v \rangle$ removes from the selection list S a node v, which is returned to the candidate list.
  Preconditions: $v \in S$, $|S| > 2$.

- Swap: the move $\mathsf{Swap}\langle v, u \rangle$ inserts a node v from the candidate list into the selection list S, and removes from the selection list a node u, which is returned to the candidate list.
  Preconditions: $v \in C$, $u \in S$.

We consider also three restricted versions of the Swap neighborhood, that we call SwapR_+, SwapR_- and $\mathsf{SwapR}_\pm$. To describe them, we introduce two additional sets, the restricted selection list S_R and the restricted candidate list C_R. The restricted selection list is defined as $S_R := \{v \in S | d_S(v) = f(S)\}$. Set S_R contains all the nodes in the solution with an edge to another node in the solution with distance equal to the current minimum distance among selected nodes, that is the value of the objective function. We denote the nodes in S_R also as the bottleneck nodes, because they are bounding the value of the objective function $f(S)$. The restricted candidate list is defined as $C_R := \{v \in C | d_S(v) > f(S)\}$, that is, all the nodes in the candidate sets with minimum distance to the nodes in the selection list greater than the current objective function value.

SwapR_+ restricts the Swap neighborhood with regards to the nodes inserted in the solution, which are chosen within the restricted candidate list C_R, but allows removal from the whole selection list S. The neighborhood SwapR_- restricts the Swap neighborhood with regards to the nodes that are removed from the solution list, that are chosen only among the nodes in S_R, but allows any insertion from the whole candidate list C. Finally, $\mathsf{SwapR}_\pm$ restricts both the insertion and the removal to the restricted lists.

We point out that also Lu et al. (2023) use a restricted version of the Swap neighborhood, but with relevant differences from us. First of all, we employ all the neighborhood variants Swap, SwapR_+, SwapR_- and $\mathsf{SwapR}_\pm$, while they only employ $\mathsf{SwapR}_\pm$. Additionally, our definition of the restricted candidate set C_R is different, as we employ a larger set, that embraces all nodes $v \in C$ with $\min_{u \in S} d_{uv} > \min_{u \in S} d_S(u) = f(S)$. The restricted selection list instead is identical. Therefore, all our neighborhood variants are larger than the Swap proposed by Lu et al. (2023).

The main reason why we can afford to use larger neighborhoods is that the metaheuristic that guides the search in our method is Simulated Annealing, which employs stochastic move selection. Lu et al., on the other hand, employ Tabu Search, which performs a complete exploration of the neighborhood at every move selection and therefore might foresee its performances reduced by the use of larger neighborhoods.

7.3.4 Move selection

We now describe how we select a random move from our Multi-Neighborhood $\mathsf{Insert} \cup \mathsf{Remove} \cup \mathsf{Swap}$. We define three real-valued parameters σ_I, σ_R, and σ_S, with $\sigma_I + \sigma_R + \sigma_S = 1$. They represent the probability of selecting the Insert, Remove, and Swap neighborhoods, respectively.

Given that, however, we have four versions of the Swap neighborhood, we define two internal biases: $b_+ \in [0,1]$ and $b_- \in [0,1]$. If the Swap neighborhood is selected, then with probability b_+ we use the restricted candidate list C_R to select the candidate node for insertion, and with probability b_- we use the restricted selection list S_R to choose the node that is removed. Therefore, we can derive that the probability of performing a regular Swap is $\sigma_S(1-b_+)(1-b_-)$, the probability of selecting SwapR_+ is $\sigma_S b_+(1-b_-)$, the probability of selecting SwapR_- is $\sigma_S(1-b_+)b_-$ and, finally, $\text{SwapR}_\pm$ is selected with probability $\sigma_S b_+ b_-$.

Now that every neighborhood is assigned a probability, we can perform a random selection of a move in two steps (three steps for Swap). First, we select the neighborhood, with a biased random selection that depends on the fixed probabilities σ_I, σ_R, and σ_S. If Swap is chosen, with another biased random selection we choose whether to use the restricted or the full candidate and solution lists, with probabilities b_+ and b_-, respectively. Finally, we draw the specific move, with a uniform selection inside the chosen neighborhood.

7.3.5 Cost function

One issue with the max-min objective function is that very large plateaus might be encountered. In fact, its value is determined by the edges in E_R. If $|E_R| = 1$, then the value is given by a single pair of nodes and to improve it suffices to remove one of the two nodes (provided that the move does not violate the capacity constraint).

When $|E_R| > 1$, unless there is a node $v \in R_S$ with $|d_S(v)| = |E_R|$, all three neighborhoods are on a plateau with regard to $f(S)$, given that all possible moves are sideways or worsening.

This situation makes the search blind about what are the best moves to perform, and it might be stuck roaming a long time on the plateau. However, we know that to improve the objective function we have first to reduce the number of edges in E_R, until we have one single edge that is binding $f(S)$, that we can finally remove and jump to a new objective function value.

Therefore, in order to deal with this situation we introduce an auxiliary cost component that guides the search on the plateaus towards solutions with smaller $|E_R|$. Given that this is less important than the actual objective, we define a lexicographic objective function as follows: given two solutions S_1 and S_2, we say that S_1 is lexicographically better than S_2 if

$$f(S_1) > f(S_2) \vee \big(f(S_1) = f(S_2) \wedge |E_{R_1}| < |E_{R_2}|\big) \tag{7.6}$$

In order to use this lexicographic function inside our metaheuristic, we

linearize it as follows:

$$f^{lex}(S) = w_{of} f(S) - |E_R| \tag{7.7}$$

with w_{of} a constant weight, assigned with a suitably high value to ensure that the second component is only relevant if two solutions have the same objective function value. The new term is subtracted and not added because the objective of the problem is to maximize the objective function, while we want to minimize $|E_R|$.

Our lexicographic objective function provides a trajectory for executing a successful sequence of moves on plateaus, so that the need for more complex neighborhoods, such as the 2-out-1-in and 1-out-2-in neighborhoods proposed by Mladenović et al. (2022), is counterbalanced.

Finally, as we mentioned in Section 7.3.1, we also allow the exploration of the infeasible region of the search space. Therefore, we also add a penalty term for capacity violations, and we weight it with a constant w_c. Again, given that the problem is a maximization one, we subtract the penalty term. We obtain the following expression to determine the cost function F used in the metaheuristic:

$$F(S) = w_c \min\{0, \sum_{v \in S} c_v - B\} + w_{of} f(S) - |E(S)| \tag{7.8}$$

The redundant data structures that we introduced in Section 7.3.1 are used to speed up the evaluation of the differential cost ΔF between two solutions S_1 and S_2, without computing $F(S_1)$ and $F(S_2)$ from scratch.

7.3.6 Metaheuristic

The metaheuristic that guides the search is the Multi-Neighborhood Simulated Annealing (MNSA) presented in Chapter 2.

7.4 Datasets and generators

Recent papers in the literature on CDP have used four datasets, called GKD-b, GKD-c, SOM-a, and MDG-b, and composed by 20, 10, 10, and 10 instances, respectively. The number of nodes is fixed, equal to 50/150 (10 each), 500, 50, 500, for all instances of the four datasets.

These datasets[1] were initially proposed in various works and then collected by Martí et al. (2010). They were originally defined for problems

[1] Available at `https://www.uv.es/rmarti/paper/mdp.html`

without the capacity constraints, therefore they have been completed by Peiró et al. (2021) by adding both the node capacities and the capacity threshold. In this process, each dataset has been duplicated into two groups by considering a threshold equal to either 0.2 or 0.3 of the sum of the capacities of the nodes. Therefore, we eventually have eight groups of instances named by adding to suffix 2 or 3 to the original name: GKD-b2, GKD-b3, GKD-c2, ...

All datasets are artificial. In detail, the datasets GKD-b and GKD-c are created by sampling points on a square and getting the Euclidean distance among all pairs of nodes. Conversely, for SOM-a and MDG-b distances are integers uniformly sampled in the range [0,9] and [0,1000], respectively.

For all instances of datasets GKD-b, GKD-c, and SOM-a, the MILP model by Martí et al. (2021) finds the optimal solutions in a few seconds. In addition, all metaheuristic methods (including ours) find quite consistently the optimal value. The comparison on these datasets therefore could be based only on the number of times that the optimal value is obtained and the time to reach it.

For the above reasons, these instances are easy and we believe that they do not represent a good benchmark anymore. We decided to discard them and to focus on the 20 instances of dataset MDG-b (10 of group MDG-b2 and 10 of group MDG-b3). We use the others only for validating the implementation of the MILP model.

The dataset MDG-b is instead quite challenging, and therefore it is a good benchmark for the comparison of optimization techniques. Nonetheless, in our opinion, it has some severe limitations:

- It is too small. Indeed, 20 instances are not enough for an extensive comparison.
- It is too homogeneous, as all instances have the same size and they are duplicated with only the capacity changed.
- Distance values are too random. In fact, they do not even satisfy the triangular inequality.
- The presence of various zeros in the distance matrix is also rather strange, given that they should represent physical distances.

For these reasons, we believe that it is necessary to introduce a new dataset so as to enrich and diversify the current benchmark. Therefore, we design a generator, based on real geographical data, that aims at creating

instances that are challenging, realistic, and diverse. Our generator, written in R language, employs the JRC-GEOSTAT 2018 population grid, that covers 38 European countries, including all the European Union Member States plus a few neighboring countries. We use the shapefile under the coordinate system EPSG:3035 with a resolution of $1km^2$, which means that it is divided in squared cells with side $1km$. Every cell contains the coordinate references and the data of the population living in the cell, updated to the year 2018, which is the most recent validated version of the grid.

The first step consists in cutting a portion of the grid based on a center (x, y) and a radius r, which implies an area of πr^2, minus the potential portion that lies on the sea. To generate a instance with size $|V|$, on this area, we select $|V|$ cells at random, through a biased random selection, with the probability proportional to the population of the cell.

Said A the set of the cells in the area, and p_n the population of the n-th cell, the probability of selecting cell n is

$$P(n) = \frac{p_n}{\sum_{m \in A} p_m} \tag{7.9}$$

After every cell selection, we sample, with a uniform random distribution, the exact coordinates of a point (x_i, y_i) within the $1km^2$ cell boundaries. Therefore, every node in the instance corresponds to a precise point in the map.

After we have sampled the $|V|$ points in the space, the procedure computes the distance matrix $|V| \times |V|$. As distances, we set the time needed to cover the routes between points by car, expressed in minutes. For this, we employ the Open Source Routing Machine, which returns the most time-effective route between two nodes. We point out that a real road distance matrix is not symmetric because of factors such as one-way directions or steepness. However, given that the problem requires a symmetric matrix, we only compute d_{ij} for $i < j$, and then we assign $d_{ji} = d_{ij}$.

In our generation procedure, the cell population is used not only to bias the sampling of the points, but it is also assigned as capacity of the node: if the node i was extracted from cell n, then $c_i = p_n$. The minimum capacity requirement is computed as a random value $B \in (B_{min}, B_{max}]$. Hereby, we set $B_{min} = max_{i \in V}(c_i)$ and $B_{max} = \min\{0.3 \sum_{i \in V} c_i, 0.9 \sum_{m \in A} p_m\}$. These values ensure that the capacity requirement B is always less than 30% of the total capacity, or the 90% of the population if this is less than the total capacity. The radius r is chosen at random in the interval $[r_{min}, r_{max}]$, with

$r_{min} = 5km$ and $r_{max} = 90km$. Finally, the possible centers (x, y) are taken from a pool of various cities and rural areas.

As a possible real-world meaning of this construction, consider that we want to decide where to place drugstores. The capacity of the node, which is the population living in the $1km^2$ where it is situated, is an approximation of the number of people placed at walking distance within the facility, which we want to satisfy by at least B. On the other hand, we want to avoid that two drugstores are placed too near, therefore we desire to maximize the minimum distance between stores. The same reasoning can be applied to other contexts, such as the placement of schools or public offices, that is convenient to build in dense areas to reduce commuting times and the use of private vehicles, while avoiding to place them too near to each other to ensure that their zones of influence don't overlap.

Using our generator, we have created a dataset composed of 200 instances with size $|V|$ between 300 and 1500, that we call GIS (give that it comes from geographical data). We use 30 instances (chosen at random, with no specific choice criterion) as validation instances and the rest as training ones. We also tested all of them with our MILP model presented in Section 7.2. We verified that no GIS instance could be solved to optimality within the one hour time limit, for which they appear to be rather challenging. All the GIS instances are available at `https://github.com/iolab-uniud/cdp`, distinguished into training and validation ones.

Table 7.1 shows the features of the validation instances. The name of the city just indicates the city or town chosen as center of the area. Sometimes, if the radius large, other cities might be included. The population is reported in thousands.

7.5 Experimental analysis

In this section we present our experimental methodology and our results, and we provide graphics and insights on the behavior of our algorithm as well as on the characteristics of the new GIS instances in relation to the algorithm performances. Our code is implemented in C++ and compiled using g++ (v. 11.4.0) in `-O3` mode. The experiments were run on a AMD Ryzen Threadripper PRO 3975WX 32-Cores (3.50 GHz) with Ubuntu Linux 22.04.3. Also the MILP model is implemented in C++, through the CONCERT interface for CPLEX (v. 22.1). The time limit for the model is set to 2880s, corresponding approximatively to one hour on the CPU used by Martí et al. (2021). The Simulated Annealing does not run on a fixed

Table 7.1: Features of the validation instances.

#	GeoData				Size			Distances					
	city	r	A	pop(k)	$\|V\|$	B(%)	B	avg	min	1st	med	3rd	max
1	Bilbao	28	2197	1131	627	0.11	918402	22.2	1	13	20	30	74
2	Bremen	51	8161	1735	1335	0.20	693367	41.7	1	27	41	54	136
3	Cagliari	50	6318	678	565	0.19	421770	39.7	1	19	38	57	159
4	Catania	51	5468	1390	714	0.24	870107	40.4	1	20	39	58	149
5	Cuneo	34	3613	549	479	0.29	196562	42.7	1	30	43	56	104
6	Exeter	77	12464	1912	1259	0.18	552502	73.4	1	46	69	96	229
7	Foggia	36	4021	409	701	0.04	177467	36.4	1	24	37	49	118
8	Fontenay	86	21804	1822	974	0.16	180151	85.2	1	59	87	111	215
9	Gottingen	29	2617	415	1206	0.08	164395	29.9	1	20	30	39	81
10	Gottingen	88	24309	3803	1358	0.05	129498	90.0	1	60	90	118	211
11	Jena	8	193	114	757	0.01	40680	11.6	1	8	11	15	45
12	Lublin	44	6073	869	1042	0.03	97965	38.9	1	22	39	53	125
13	Milano	32	3205	4992	568	0.24	909420	31.1	1	21	30	40	84
14	Milano	53	8797	7612	692	0.27	906640	44.6	1	30	43	57	131
15	Munich	7	145	988	1343	0.05	697453	12.6	1	9	12	16	33
16	Munich	90	25433	6024	798	0.16	463017	60.8	1	40	59	79	179
17	Napoli	57	7158	5222	1346	0.23	2124227	45.8	1	25	40	58	292
18	Nowemiasto	89	24833	2104	1487	0.03	122757	104.8	1	68	104	141	247
19	Palermo	88	12199	2159	1186	0.06	369869	81.7	1	40	83	117	222
20	Pamplona	23	1649	379	512	0.05	251664	11.1	1	7	10	13	76
21	Rome	15	697	2607	700	0.19	1344222	19.9	1	14	20	26	49
22	Sofia	47	6917	1608	1034	0.18	1237857	28.7	1	13	21	39	200
23	Split	41	4412	390	614	0.04	134182	40.0	1	11	28	48	264
24	Suzzara	68	14493	3822	1447	0.27	947949	71.9	1	49	73	95	170
25	Tampere	19	1125	344	664	0.16	278974	18.7	1	13	18	24	68
26	Ulm	54	9141	1903	1241	0.19	411900	53.7	1	37	54	70	133
27	Ulm	8	193	191	845	0.04	124649	10.9	1	8	11	14	31
28	Vienna	53	8797	3083	1469	0.10	1207533	32.1	1	17	29	45	121
29	Wroclav	48	7209	1302	1256	0.02	107873	39.5	1	22	39	55	129
30	Zagreb	53	8797	1528	980	0.15	600667	39.1	1	20	38	55	141

Table 7.2: Parameter tuning for the Multi-Neighborhood Simulated Annealing.

Name	Description	Initial Range	value MDG-b	GIS
T_0	Start temperature	[300, 2000]	194.49	168.78
T_f	Final temperature	[0.01,10.00]	0.3758	0.0593
w_s	Weight of $f(S)$ in cost function F	[1,100]	11	90
σ_I	Rate of Insert neighborhood	[0.00,0.25]	0.131	0.081
σ_R	Rate of Remove neighborhood	[0.00,0.25]	0.129	0.009
b_+	Bias toward C_R	[0.00,1.00]	0.449	0.320
b_-	Bias toward S_R	[0.00,1.00]	0.359	0.593

runtime but on a fixed number of iterations, that was 50 millions for the tuning phase and 100 millions for the validation. Except when explicitly stated, one single core was dedicated to each experiment.

7.5.1 Tuning

We tuned our solver using IRACE, which is a tool for automatic algorithm configuration based on iterated racing (López-Ibáñez et al., 2016). Given that the datasets MDG-b and GIS have different features, we performed two separated tuning procedures.

Specifically, the tuning was done only in the training instances for the GIS dataset, whereas it has been done on all instances for MDG-b, because there are no dedicated instances available for training purposes.

The tuning was carried out in two stages. In the first stage, we tuned the temperatures of Simulated Annealing and the weight w_{of}. In the second stage, we tuned the neighborhood rates and biases. The value w_c is preventively fixed to 10000. Also the values of the cooling rate α and the cut-off ρ are fixed to 0.988 and 0.16, respectively. These values where given by preliminary experiments, that showed that the results are quite insensible to these parameters. The advantage of not including them in the final tuning procedure is to reduce the parameter space and to get higher precision on more significant parameters, like initial or final temperature. Regarding the neighborhood rates, we only tune σ_I and σ_R, because σ_S is computed as $1 - \sigma_I - \sigma_R$. Table 7.2 shows the results of the tuning procedure, separately for MDG-b and GIS dataset.

7.5.2 Results

Table 7.3: Results of the MILP models aggregated by dataset.

	Instance		Martí et al. 2021			Peiró et al. 2021			Our Model		
B	Fam.	$\|V\|$	LB	UB	time	LB	UB	time	LB	UB	time
	GKD-b2	50	112.3	112.3	0.1	112.3	112.3	2.0	112.33	112.33	5.4
	GKD-b2	150	118.7	118.7	0.6	118.7	118.7	669.9	118.73	118.73	160.9
0.2	GKD-c2	500	9.4	9.4	5.7	6.8	20.5	3600.2	7.82	22.92	3596.3
	SOM-a2	50	4.1	4.1	0.0	4.1	4.1	1.3	4.10	4.10	7.2
	MDG-b2	500	0.0	125.0	3643.5	11.5	973.8	3600.2	34.50	1000.00	3596.2
	GKD-b3	50	97.8	97.8	0.0	97.8	97.8	2.5	97.79	97.79	11.8
	GKD-b3	150	108.1	108.1	0.3	108.1	108.2	1519.5	108.11	108.11	243.6
0.3	GKD-c3	500	8.4	8.4	1.4	4.9	22.8	3600.1	6.30	22.92	3596.2
	SOM-a3	50	2.1	2.1	0.0	2.1	2.1	1.0	2.10	2.10	11.3
	MDG-b3	500	3.1	60.2	3650.9	0.9	992.0	3600.1	15.48	1000.00	3596.6

Table 7.3 shows the results obtained by our model. We report the average solution found, the upper bound, and the running time (rescaled to 1h), that might be less than the time limit in case the optimal solution is found. What we observe is that all models find the optimal solutions on all instances with up to 150 nodes. The model from Martí et al. solves to optimality in few seconds also the instances from the class GKD-c. Therefore, the only challenging instances are the MDG-b. On those instances, our model shows a better performance than the other models, but, differently from Martí et al., it is not capable of finding good bounds. In Table 7.4 we report the detailed results obtained by our model on MDG-b instances. Furthermore, we report the results obtained by the model running on 16 cores, in parallel, for 12 hours. The reason of this test was to understand if MDG-b instances are out of reach or if the model could find optimal solutions within a longer time limit. The results are good, but not proven optimal. Except for two instances, that are highlighted in bold, they are below the best results obtained by the metaheuristics in much short time. Therefore, we confirm that MDG-b is the only challenging dataset.

Table 7.5 shows the results obtained by 40 repetitions per instance of our solver on the MDG instances. We compare our results with those from Lu et al. (2023) and Mladenović et al. (2022), obtained respectively on 40 and 20 repetitions per instance. We do not report other results from the literature, because they lag quite behind. Following the line set by both Lu

Table 7.4: Model results on MDG-b dataset.

instance	1h		12h × 16vCPU	
	distance	upper bound	distance	upper bound
MDG-b_01_n500_b02_m50	33.6	1000.0	**64.6**	861.2
MDG-b_02_n500_b02_m50	28.9	1000.0	58.5	919.1
MDG-b_03_n500_b02_m50	20.5	1000.0	58.8	918.1
MDG-b_04_n500_b02_m50	32.8	1000.0	54.7	914.8
MDG-b_05_n500_b02_m50	50.3	1000.0	54.2	912.8
MDG-b_06_n500_b02_m50	34.6	1000.0	57.5	917.2
MDG-b_07_n500_b02_m50	34.9	1000.0	52.5	922.5
MDG-b_08_n500_b02_m50	35.0	1000.0	57.2	882.2
MDG-b_09_n500_b02_m50	41.9	1000.0	59.2	919.6
MDG-b_10_n500_b02_m50	32.5	1000.0	**63.0**	906.9
MDG-b_01_n500_b03_m50	13.9	1000.0	27.5	829.7
MDG-b_02_n500_b03_m50	14.5	1000.0	27.4	844.2
MDG-b_03_n500_b03_m50	13.7	1000.0	28.7	843.9
MDG-b_04_n500_b03_m50	17.8	1000.0	29.9	843.2
MDG-b_05_n500_b03_m50	14.5	1000.0	29.2	847.5
MDG-b_06_n500_b03_m50	15.1	1000.0	28.9	856.8
MDG-b_07_n500_b03_m50	15.6	1000.0	29.6	831.5
MDG-b_08_n500_b03_m50	19.9	1000.0	31.6	844.2
MDG-b_09_n500_b03_m50	16.2	1000.0	28.7	835.1
MDG-b_10_n500_b03_m50	13.6	1000.0	30.4	824.1

et al. and Mladenović et al., beside the average we also report the best and worst solution found for each instance in the batch of 40 runs. Furthermore, we report the average running time, because our solution method works on a fixed number of iterations, which cause the running time to change from run to run.

Lu et al. and Mladenović et al. used a fixed time limit of 300s and 60s, respectively, so we do not report it in the table. Lu et al. also solve the instances on a shorter time limit of 10s, but they get worse results, that we don't report. Finally, we also report in the last three columns the delta between our results and the best between the other two solvers, separately for the best, average and worst solution.

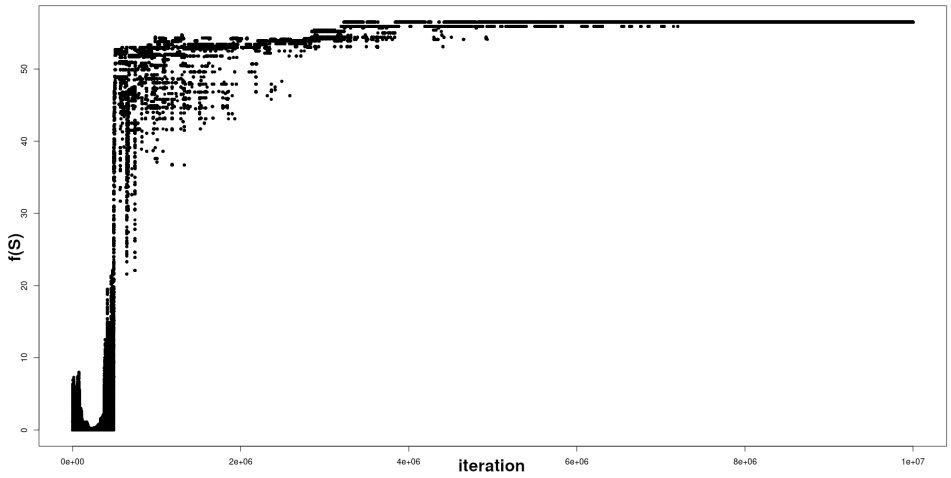
The outcome is that our solver finds the best average solution on 19 out of 20 instances, and only on the instance MDG-b_01_n500_b03_m50 it performs worse than Mladenović et al., by just 0.05. For all the other instances we improve the previous results, with an average delta that goes from as little as 0.02 on the instance MDG-b_10_n500_b02_m50 to as

Table 7.5: Results comparison between us and benchmark.

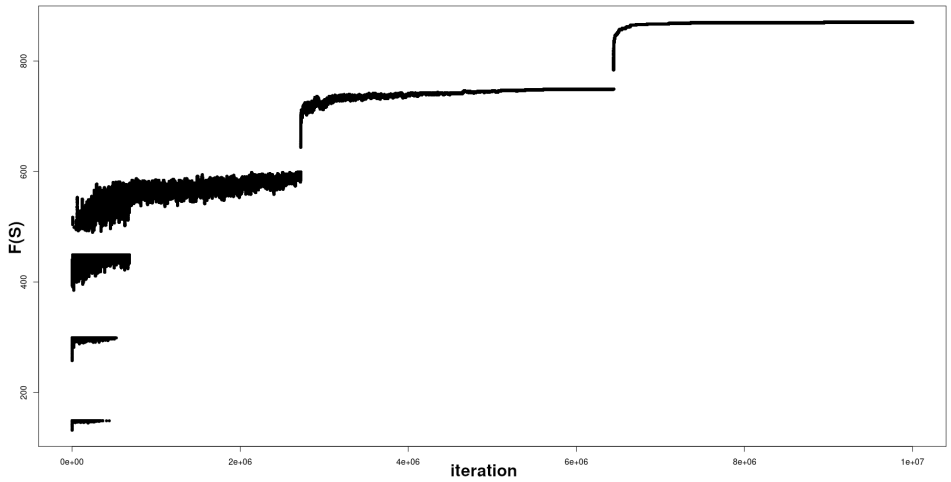
Inst.		Lu et al. 2023			Mladenović et al. 2022			MNSA				Difference		
B	#	best	avg	worst	best	avg	worst	time	best	avg	worst	best	avg	worst
0.2	1	64.6	61.5	50.8	64.60	**63.04**	62.10	43.35	64.60	62.99	61.90	0.00	-0.05	-0.20
0.2	2	60.8	56.3	51.6	60.40	59.42	58.70	43.47	60.80	**60.22**	58.70	0.00	0.80	0.00
0.2	3	60.9	59.8	54.8	60.90	60.59	59.90	42.26	60.90	**60.80**	57.80	0.00	0.21	-2.10
0.2	4	56.6	54.4	48.8	56.90	55.75	54.80	44.27	57.40	**56.52**	54.90	0.50	0.77	0.10
0.2	5	58.2	54.0	51.0	58.20	57.23	57.10	42.48	58.90	**58.63**	57.10	0.70	1.40	0.00
0.2	6	60.6	55.7	49.4	60.60	58.38	56.60	42.06	60.70	**59.77**	57.20	0.10	1.39	0.60
0.2	7	59.3	54.0	47.9	58.00	55.40	53.40	44.78	59.30	**58.17**	54.80	0.00	2.77	1.40
0.2	8	60.4	56.2	48.5	60.10	59.46	58.60	42.22	61.10	**60.09**	59.70	0.70	0.63	1.10
0.2	9	60.2	57.5	52.6	59.70	58.10	57.00	42.55	60.90	**60.10**	58.60	0.70	2.00	1.60
0.2	10	63.0	61.3	52.7	63.00	62.48	62.00	44.03	63.00	**62.50**	62.00	0.00	0.02	0.00
0.3	1	27.2	24.2	19.6	28.70	27.73	27.30	54.94	29.00	**28.53**	27.90	0.30	0.80	0.60
0.3	2	27.6	25.5	21.5	28.20	27.73	27.30	51.57	29.10	**28.78**	27.70	0.90	1.05	0.40
0.3	3	30.0	28.0	24.9	29.40	28.61	28.10	52.30	30.10	**29.96**	29.60	0.10	1.35	1.50
0.3	4	28.0	25.3	23.2	29.70	29.07	28.90	53.10	30.20	**29.86**	29.50	0.50	0.79	0.60
0.3	5	27.9	25.3	23.0	29.70	28.69	28.30	52.54	30.00	**29.64**	29.20	0.30	0.95	0.90
0.3	6	28.2	26.0	22.8	28.90	28.28	28.00	51.34	29.00	**28.89**	28.60	0.10	0.61	0.60
0.3	7	28.4	26.0	20.3	29.40	28.54	28.00	54.58	30.50	**30.20**	29.50	1.10	1.66	1.50
0.3	8	31.1	28.3	24.2	31.90	31.73	31.10	49.17	31.90	**31.84**	31.70	0.00	0.11	0.60
0.3	9	29.7	25.9	20.8	29.70	28.88	28.20	50.19	29.80	**29.75**	28.70	0.10	0.87	0.50
0.3	10	29.1	26.7	21.7	30.40	29.55	29.40	50.46	30.50	**29.96**	29.40	0.10	0.41	0.00

much as 2.77 on the instance MDG-b_07_n500_b02_m50. Regarding the best solution, we always improve or match the one found by the others, and regarding the worst solution found on the batch of forty runs, we get better results on 14 instances, and we are below Mladenović et al. on two instances. Overall, we can say that our solver performs better than the previous ones from the literature, and that Mladenović et al. is the closest competitor.

Finally, we report the results obtained on the GIS dataset by our MILP model in 1h and by our solver on 40 runs. We also compiled and run on our machine the solver from Lu et al., within a time limit of 300s. According to the results, a few instances seem to be easy, as all the methods consistently get the same objective function value. Those instance are GIS-1 and GIS-11. On all instances our solution method performs better than or as equal as Lu et al. and the MILP model.



(a) Evolution of $f(S)$ on instance MDG-b_01_n500_b02_m50.



(b) Evolution of $F(S)$ on instance GIS-24, with w_{of} rescaled from 90 to 15, to let better appreciate the contribution of the $|E_R|$ cost component.

Figure 7.1: Search patterns on MDG-b and generated instances.

7.5.3 Algorithmic insights

Fig. 7.1 illustrates the evolution in the objective function value over the course of a short execution (10 millions iterations) of our solver on a MDG-b instance and on a GIS instance. The y-axis represents the distance value of the incumbent solution, and the x-axis shows the iteration count.

In Fig. 7.1a, we see that the objective function exhibits significant oscillations, and it is only towards the end that stabilizes around the final value. This is different to what we observe in Fig. 7.1b, relative to one of the GIS instances. For better understanding this case, we show the evolution of the cost function that we use in our solution method in Eq. (7.8), that include the lexicographic objective function. From the graphic, we can appreciate the fact that the metaheuristic explores the plateau only thanks to the gradient offered by the auxiliary cost component. There are also fewer jumps among different minimum distance values because of the large numbers of bottleneck edges. We can also appreciate that the smaller scale of the bottleneck edge component prevents it from interfering the with the actual objective.

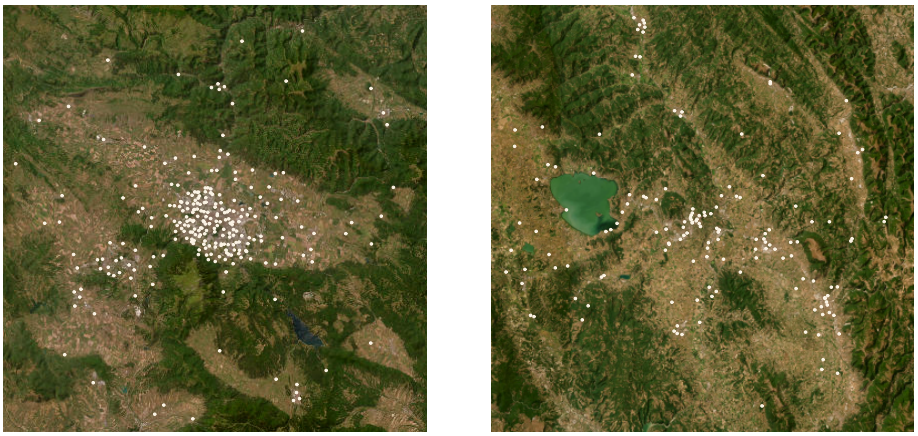


Figure 7.2: Example of solutions on different urban areas. Left: Sofia, Bulgaria, $|V| = 1034$, $r = 47\ km$, $B = 18\%$; Right: Perugia, Italy, $|V| = 597$, $r = 43\ km$, $B = 4\%$; Maps are courtesy of ESRI.

Fig. 7.2 is aimed to illustrate the geographical dimension of the new instances. It shows two plots of the solutions obtained by our method on two different generated instances. We can observe than on the city on the right (Sofia, Bulgaria, GIS-22) the population concentration in the city together with a rather high capacity requirement (18%) makes it hard to sparsify the

points. On the other hand, the instance on the right (Perugia, Italy, from the training set) has a lower capacity requirement and the more sprawled population distribution makes it easier to sparsify the points over the map, even though a certain degree of concentration in denser portions of the map remains unavoidable.

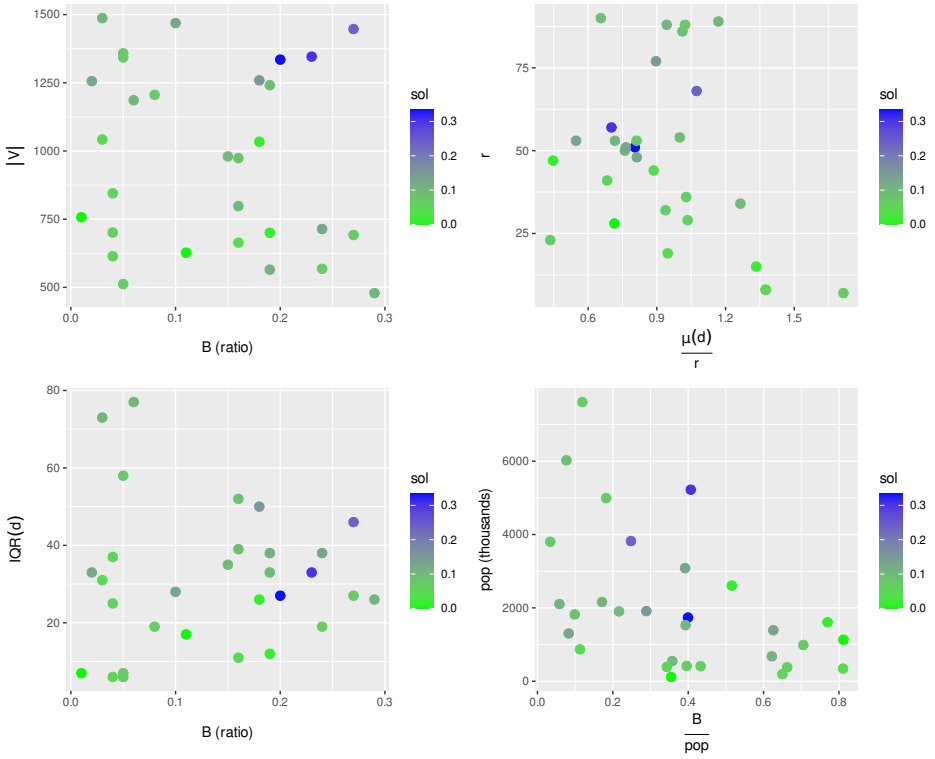


Figure 7.3: Instance features and the corresponding average difference solution between solvers (color).

Finally, Fig. 7.3 shows the pairwise distribution of different instance features for the 30 validation instances of dataset GIS. Besides the size of the graph $|V|$, we consider the relative capacity requirement B, the radius r, the ratio between the average distance and the radius $\mu(d)/r$, the interquartile range of the distances $IQR(d)$, the population in the area pop, and the ratio between the required capacity and the population B/pop.

We can appreciate how all the selected features distribute evenly along their domains, that was one of the goals of our generator. The color of the points indicates the average relative difference between the performance of

the three solvers. There are some instances that present higher variability in the performance of the solvers, but no clear correlation on the features emerges.

7.6 Conclusions

We proposed a multi-neighborhood approach guided by Simulated Annealing for the capacitated dispersion problem. The outcome has been that our approach, properly engineered and tuned, is able to outperform the current best results in the literature on almost all instances.

We also proposed a new dataset that in our opinion has all the features it takes to become a new benchmark for this problem. At the same time, we propose to dismiss four out of five of the current datasets, as they are not challenging anymore, and their perpetuation brings a waste of work and computational time.

Finally, we developed and implemented a new mathematical model that is able to provide competitive results, especially on long runs.

Table 7.6: Results on GIS dataset.

	MILP (1h)		MNSA				Lu et al.		
#	LB	UB	time (s)	max	avg	min	max	avg	min
1	**6**	74	81.0	6	**6.00**	6	6	**6.00**	6
2	2	136	145.1	5	**5.00**	5	5	**5.00**	5
3	**4**	159	73.9	4	**4.00**	4	4	3.05	3
4	3	149	86.8	4	**4.00**	4	3	3.00	3
5	**5**	104	65.3	5	**5.00**	5	4	3.92	3
6	5	229	132.7	7	**7.00**	7	6	5.10	5
7	7	118	81.5	8	**8.00**	8	7	6.97	6
8	9	215	127.6	11	**10.10**	10	9	7.85	7
9	6	81	134.6	7	**7.00**	7	7	6.10	6
10	28	211	126.2	34	**33.42**	32	31	28.57	26
11	**11**	45	79.1	11	**11.00**	11	11	**11.00**	11
12	14	125	131.6	16	**15.18**	15	15	13.35	12
13	6	84	70.3	7	**7.00**	7	6	6.00	6
14	7	131	76.2	8	**8.00**	8	7	6.67	6
15	5	33	123.2	6	**5.83**	5	5	4.94	4
16	10	179	89.0	12	**11.88**	11	10	9.40	9
17	2	292	141.8	5	**5.00**	5	5	4.02	4
18	40	247	110.3	43	**42.98**	42	40	32.02	11
19	13	222	126.3	16	**16.00**	16	14	12.47	10
20	6	76	68.3	7	**7.00**	7	6	6.00	6
21	**5**	49	85.1	5	**5.00**	5	5	4.85	4
22	**4**	200	113.8	4	**4.00**	4	4	3.90	3
23	6	264	84.1	8	**7.03**	7	7	6.52	6
24	3	170	144.4	6	**6.00**	6	5	5.00	5
25	4	68	86.4	5	**4.40**	4	4	4.00	4
26	7	133	124.2	8	**8.00**	8	6	6.00	6
27	6	31	84.5	7	**7.00**	7	7	6.60	6
28	4	121	168.8	6	**6.00**	6	5	5.00	5
29	15	129	144.7	22	**20.98**	20	20	16.65	14
30	4	141	113.7	5	**5.00**	5	4	4.00	4

Part III

Applications of CMSA

Chapter 8

Bus Driver Scheduling

Driver scheduling problems are complex combinatorial problems that integrate the scheduling part with routing issues, due to the fact that drivers and vehicles get moved to different locations by their duties. Different driver scheduling problems have been proposed in the literature, differing among themselves mainly depending on the type of vehicles that are involved and constraints.

We consider here a Bus Driver Scheduling (BDS) problem, which is characterized by the fact that the atomic driving duties (called *legs*) are short compared to other vehicles (e.g., planes or trains). Therefore, the daily shift of a driver is composed of a relatively large number of independent legs, which must be assembled in a working shift respecting various regulations mainly connected to safety issues.

We focus on the specific BDS formulation proposed by Kletzander and Musliu (2020), which arises from a public transportation setting in Austria and is subject to many constraints related to rest time (breaks) regulated by legal requirements and collective agreements. This formulation comes with a challenging dataset composed of many realistic instances, which has already been used in the experimental analysis of a few exact and metaheuristic techniques (Kletzander et al., 2021; Kletzander and Musliu, 2020; Kletzander et al., 2022).

We solve the problem by means of Construct, Merge, Solve and Adapt (CMSA), seen in Chapter 3. As constructor, we use a greedy algorithm developed in a previous work (Kletzander and Musliu, 2020), that we suitably randomized in order to employ it for the generation of solutions within the CMSA algorithm.

For our CMSA solver, we performed a principled tuning procedure in

Table 8.1: A Bus Tour Example

ℓ	$tour_\ell$	$start_\ell$	end_ℓ	$startPos_\ell$	$endPos_\ell$
1	1	360	395	0	1
2	1	410	455	1	2
3	1	460	502	2	1
4	1	508	540	1	0

order to obtain the best configuration of the parameters and we compared our tuned solver with the best results from the literature. The outcome is that our solver is able to improve the state-of-the-art results for a range of problem instances, in particular for the large ones.

8.1 Problem description

The investigated Bus Driver Scheduling problem deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day of operation, according to the rules specified by an Austrian collective agreement. We use the same specification as presented in Kletzander and Musliu (2020), where the reader can find a more detailed description of the problem.

8.1.1 Problem input

The bus routes are given as a set $\mathcal{L}$ of individual bus legs, each leg $\ell \in \mathcal{L}$ is associated with a tour, denoted as $tour_\ell$ (corresponding to a particular vehicle), a start time $start_\ell$, an end time end_ℓ, a starting position $startPos_\ell$, and an end position $endPos_\ell$. The actual driving time for the leg is denoted by $drive_\ell$. We assume that $drive_\ell = length_\ell = end_\ell - start_\ell$.

Table 8.1 shows a short example of one particular bus tour. The vehicle starts at time 360 (6:00 am) at position 0, does multiple legs with stops including waiting time at positions 1 and 2 and finally returns to position 0. A valid tour never has overlapping bus legs and consecutive bus legs satisfy $endPos_i = startPos_{i+1}$. A tour change occurs when a driver has an assignment of two consecutive bus legs i and j with $tour_i \neq tour_j$.

A distance matrix specifies, for each pair of positions p and q, the time $d_{p,q}$ a driver takes to get from p to q when not actively driving a bus. If no transfer is possible, then $d_{p,q} = \infty$. $d_{p,q}$ with $p \neq q$ is called the *passive ride*

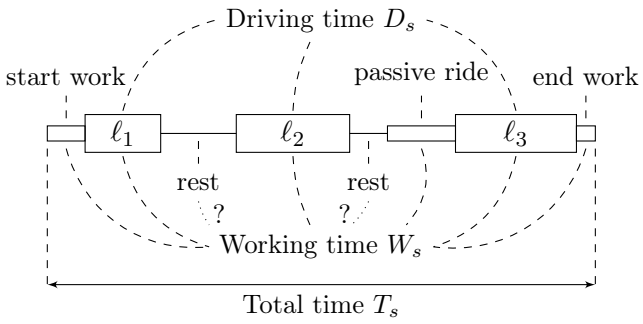


Figure 8.1: Example shift

time. $d_{p,p}$ represents the time it takes to switch tour at the same position, but is not considered passive ride time.

Finally, each position p is associated with an amount of working time for starting a shift ($startWork_p$) and ending a shift ($endWork_p$) at that position. The instances in this chapter use $startWork_p = 15$ and $endWork_p = 10$ at the depot ($p = 0$), to take into account the time needed to enter and exit the depot. These values are 0 for other positions, given that the bus is already on the street.

8.1.2 Solution

A solution to the problem is an assignment of exactly one driver to each bus leg. Criteria for feasibility are:

- No overlapping bus legs are assigned to any driver.
- Changing tour or position between consecutive assignments i and j requires $start_j \geq end_i + d_{endPos_i, startPos_j}$.
- Each shift respects all hard constraints regarding work regulations as specified in the next section.

8.1.3 Work and break regulations

Valid shifts for drivers are constrained by work regulations and require frequent breaks. First, different measures of time related to a shift s containing the set of bus legs $\mathcal{L}_s$ need to be distinguished, as visualized in Fig. 8.1:

- The total amount of driving time: $D_s = \sum_{i \in \mathcal{L}_s} drive_i$

- The span from the start of work until the end of work T_s with a maximum of $T_{max} = 14$ hours.
- The working time $W_s = T_s - \mathit{unpaid}_s$, not including certain unpaid breaks.

Driving time regulations

The maximum driving time is restricted to $D_{max} = 9$ hours. The whole distance $start_j - end_i$ between consecutive bus legs i and j qualifies as a driving break, including passive ride time. Breaks from driving need to be taken repeatedly after at most 4 hours of driving time. In case a driving break is split in several parts, all parts must occur before a driving block exceeds the 4-hour limit. Once the required amount of break time is reached, a new driving block starts. The following options are possible:

- One break of at least 30 minutes
- Two breaks of at least 20 minutes each
- Three breaks of at least 15 minutes each

Working time regulations

The working time W_s has a hard maximum of $W_{max} = 10$ hours and a soft minimum of $W_{min} = 6.5$ hours. If the employee is working for a shorter period of time, the difference has to be paid anyway. The actual paid working time is $W'_s = \max\{W_s;\ 390\}$.

A minimum rest break is required according to the following options:

- $W_s < 6$ hours: no rest break
- 6 hours $\leq W_s \leq 9$ hours: at least 30 minutes
- $W_s > 9$ hours: at least 45 minutes

The rest break may be split into one part of at least 30 minutes and one or more parts of at least 15 minutes. The first part has to occur after at most 6 hours of work. Note that a break can be a rest break and driving break simultaneously or just qualify as one of the two types.

Whether rest breaks are paid or unpaid depends on break positions according to Fig. 8.2. Every period of at least 15 minutes of consecutive rest break is unpaid as long as it does not intersect the first 2 or the last 2 hours of the shift (a longer rest break might be partially paid and partially unpaid). The maximum amount of unpaid rest is limited:

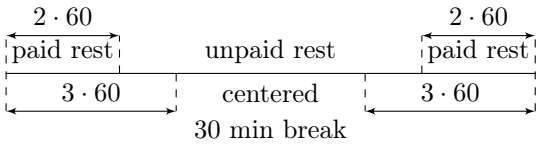


Figure 8.2: Rest break positioning

- If 30 consecutive minutes of rest break are located such that they do not intersect the first 3 hours of the shift or the last 3 hours of the shift, at most 1.5 hours of unpaid rest are allowed.
- Otherwise, at most one hour of unpaid rest is allowed.

Rest breaks beyond this limit are paid.

Split shifts

If a rest break is at least 3 hours long, it is instead considered a shift split, which is unpaid and does not count towards W_s. However, such splits are typically regarded badly by the drivers. A shift split counts as a driving break, but does not contribute to rest breaks.

8.1.4 Objectives

As argued by Kletzander and Musliu (2020), practical schedules must not consider only operation costs. The objective

$$cost_s = 2 \cdot W'_s + T_s + ride_s + 30 \cdot ch_s + 180 \cdot split_s \tag{8.1}$$

represents a linear combination of several criteria for shift s. The paid working time W'_s is the main objective and it is combined with the total time T_s to reduce long unpaid periods for employees. The next sub-objectives reduce the passive ride time $ride_s$ and the number of tour changes ch_s, which is beneficial for both employees and efficient schedules. The last objective aims to reduce the number of shift splits $split_s$ as they are very unpopular. Finally, Fig. 8.3 shows two examples of solutions of the BDS on a small and large instance.

8.2 Related work

Different variants of BDS have been studied from the early 60's (Wren, 2004). The BDS is often modelled as a Set Partitioning Problem and

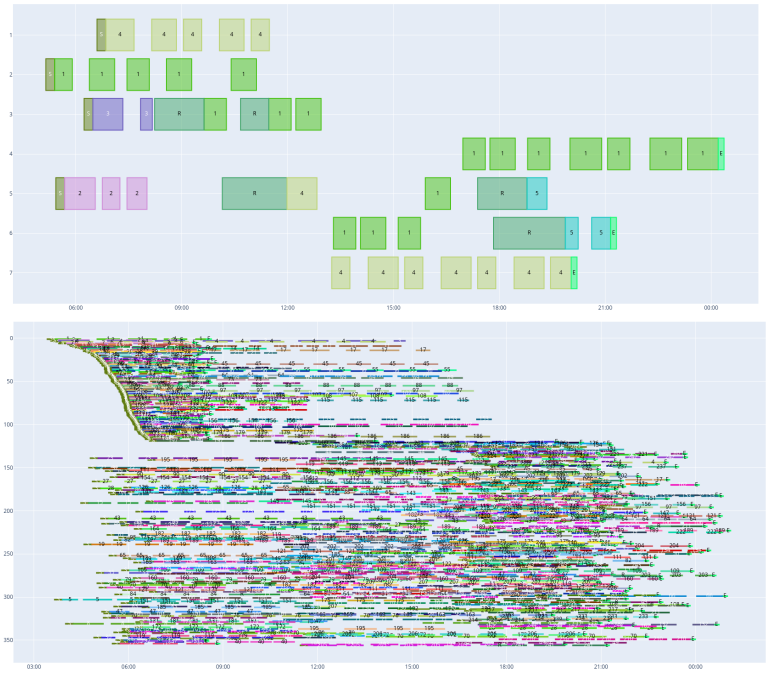


Figure 8.3: Solutions of the BDS on a small (above) and large (below) instance.

exact methods have been used in many publications to solve various variants of this problem (Smith and Wren, 1988; Desrochers and Soumis, 1989; Lin and Hsu, 2016; Portugal et al., 2008; Kletzander et al., 2021). To solve very large real-world problems in a reasonable time, several metaheuristic methods have been studied for BDS. Such methods include Greedy approaches (Martello and Toth, 1986), Tabu Search (Shen and Kwan, 2001; Kletzander et al., 2022), Simulated Annealing (Kletzander and Musliu, 2020), GRASP (De Leone et al., 2011), and Genetic Algorithms (Li and Kwan, 2003; Lourenço et al., 2001).

Bus Driver Scheduling is one of six steps in the overall process of operating bus transport systems (Ibarra-Rojas et al., 2015), located between vehicle scheduling and driver rostering. It is a variant of crew scheduling problems (Ernst et al., 2004) and it is related to problems like airline (Gopalakrishnan and Johnson, 2005) or train crew scheduling.

The problem definition of BDS is highly dependent on the country's labour regulations, therefore, algorithms for other BDS variants cannot be used directly for the Austrian rules, which are more complex than most found

in the literature. Previous work mostly focuses on cost only, sometimes including minimizing idle times and vehicle changes (Ibarra-Rojas et al., 2015; Constantino et al., 2017), but without considering the additional objectives for shift ergonomics that are considered for the BDS problem in this chapter. Our problem variant has been introduced recently in the literature, and, to the best of our knowledge, the recently introduced exact approach based on Branch and Price (Kletzander et al., 2021), the metaheuristic approaches Simulated Annealing (Kletzander and Musliu, 2020) and Tabu Search (Kletzander et al., 2022), as well as the application of problem-independent hyper-heuristics in combination with a set of problem-dependent low-level heuristics (Kletzander and Musliu, 2022), represent the current state of the art for this problem. Although these approaches give very good results, the optimal solutions are still not known for most instances. Therefore, the investigation of new approaches is important for this problem.

8.3 The CMSA approach to the BDS problem

Our CMSA algorithm for the BDS Problem is based on the following main idea. Given the set of legs $\mathcal{L} = \{\ell_1 ... \ell_n\}$, let $\mathcal{C}$ be the collection of all possible feasible bus shifts, where each shift $s \in \mathcal{C}$ is a sequence of legs that does not violate any of the constraints of the problem. A feasible solution is any collection of shifts $\phi \subset \mathcal{C}$ such that every leg $\ell \in \mathcal{L}$ belongs to one and only one shift $s \in \phi$. Solution ϕ is a valid solution for the set partitioning problem on $\mathcal{C}$. Let then $t_{\ell s} \in \{0, 1\}$ be 1 if leg ℓ forms part of shift s, and 0 otherwise. Moreover, let c_s be the cost of shift s, calculated according to the objectives explained in Section 8.1. If we were able to enumerate all shifts in $\mathcal{C}$, the optimal solution of the BDS problem could be found by solving the following ILP model of the set partitioning problem to optimality.

$$\min \sum_{s \in \mathcal{C}} c_s x_s \tag{8.2}$$

$$\text{s.t.} \sum_{s \in \mathcal{C}} x_s t_{\ell s} = 1 \qquad \forall\, \ell \in \mathcal{L} \tag{8.3}$$

$$x_s \in \{0, 1\} \qquad \forall\, s \in \mathcal{C} \tag{8.4}$$

This ILP model is based on a binary variable x_s for each bus shift $s \in \mathcal{C}$, whereby a value of $x_s = 1$ means that shift s is chosen to be part of the solution. Moreover, Eq. (8.3) ensure that each leg ℓ in $\mathcal{L}$ is present exactly once among the chosen bus shifts. In this way, all bus legs will be assigned

to exactly one bus driver and no legs will be left uncovered. The objective at Eq. (8.2) is to minimize the total cost, which is the sum of the costs c_s of the shifts that belong to the solution.

Nonetheless, in real-world instances, and in most instances proposed for this formulation, the cardinality of set $\mathcal{C}$ is too big for making the enumeration of the shifts a practical solution, and even the application of some efficient generation procedures, such as backtracking, would lead to ILP models that are too large to be solved in reasonable time with the current availability of memory and computational resources. While we cannot solve the set partitioning problem on $\mathcal{C}$, we can use the above ILP model for solving the reduced sub-instances $\mathcal{C}' \subset \mathcal{C}$, as required by the solve phase of CMSA.

The CMSA employed is the classical CMSA presented in Section 3.1. The construction procedure is a randomized version of the greedy originally proposed by Kletzander and Musliu (2020). It is described in Section 8.3.1.

Our CMSA takes as input the values for the following three parameters:

- n_{sols}, which fixes the number of solutions to be probabilistically generated by the construction procedure at each CMSA iteration.
- d_{rate}, which guides the determinism rate in the solution construction procedure.
- age_{limit}, which limits the number of iterations a solution component (shift) s can remain in the sub-instance $\mathcal{C}'$ without being chosen by the exact solver. Note that the age of a solution component s is maintained in a variable $age[s]$.

8.3.1 Greedy heuristic

The greedy heuristic employed in the construction step of our CMSA, which is called in the CONSTRUCT phase of CMSA, is described in Fig. 8.4. It is a revisited version of the greedy algorithm proposed in Kletzander and Musliu (2020), suitably randomized for the purpose of working as constructor within CMSA. The procedure takes as input a value for parameter d_{rate}. The algorithm starts by sorting the legs, which is done at line 3 of Fig. 8.4 in function ApplySorting. This sub-procedure adds the legs–one by one–into a sorted sequence $\mathcal{L}_{sorted}$, initially empty, choosing among those legs that have not been added to $\mathcal{L}_{sorted}$ yet. Every new entry is chosen according to the following criterion: with probability d_{rate}, the leg with the earliest start time is added to $\mathcal{L}_{sorted}$. Otherwise–that is, with probability $1 - d_{rate}$–a random leg is chosen. If d_{rate} is set to 1.0, legs in $\mathcal{L}_{sorted}$ are sorted according to

their start time, as done in the original algorithm (Kletzander and Musliu, 2020). Then, beginning at line 4, the main loop of the algorithm takes place. The legs are explored in the order defined by $\mathcal{L}_{sorted}$ and each leg ℓ is inserted either in the shift that produces the least cost increase or a new shift is created, if the cost of the new shift containing solely ℓ is less than the least cost increase plus a certain threshold τ. Function SetThreshold chooses the value of τ as follows: with probability d_{rate}, τ is set to a fixed value of 500, while with probability $1 - d_{rate}$, a random number between 500 and 1000 is chosen uniformly. These bounds (500, respectively 1000) were selected according to problem-specific knowledge. After inserting a leg ℓ in an existing or in a new shift, the algorithm tries to perform all feasible additions of other legs ℓ' that belong to the same tour of ℓ to that shift. This sub-procedure explores the legs by increasing start time, and it terminates at the first infeasible insertion or where no other legs with the same tours are left. The procedure ends when all legs from $\mathcal{L}_{sorted}$ have been added to the shifts in the solution Φ_{cur}.

procedure PROBABILISTIC GREEDY PROCEDURE(Set of legs $\mathcal{L}$, value for d_{rate})
1: $\Phi_{cur} \leftarrow \emptyset$
2: $\mathcal{L}_{sorted} = \mathsf{ApplySorting}(\mathcal{L}, d_{rate})$
3: **for** all ℓ in $\mathcal{L}_{sorted}$
4: $s_{best} = \text{argmin}_{s \in \Phi_{cur}}(c_{s \cup \{\ell\}} - c_s)$
5: $\tau = \mathsf{SetThreshold}(d_{rate})$
6: **if** $c_{\{\ell\}} < c_{s_{best} \cup \{\ell\}} - c_{s_{best}} + \tau$
7: Add new shift $\{\ell\}$ to Φ_{cur}
8: **else**
9: add leg ℓ to shift s_{best} in Φ_{cur}
10: **for** all $\ell' \neq \ell$ in $\mathcal{L}_{sorted}$ such that $tour(\ell') = tour(\ell)$
11: add leg ℓ' to shift s_{best} in Φ_{cur} if $s_{best} \cup \{\ell'\}$ is feasible
12: remove from $\mathcal{L}_{sorted}$ all legs added to shifts in Φ_{cur}.
13: **return** Φ_{cur}

Figure 8.4: Probabilistic greedy procedure

8.3.2 Sub-instance and exact solver

The sub-instance is the collection $\mathcal{C}' \subset \mathcal{C}$ of shifts generated by the construction procedure. The model solved in the SOLVE phase is the weighted set partitioning described in Eq. (8.2), on the sub-instance $\mathcal{C}'$.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
c_s	9	8	1	3	4	9	5	3	2	1	5	8	2	4	6
ℓ_1	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0
ℓ_2	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
ℓ_3	0	1	0	0	0	1	0	0	0	1	1	1	0	1	0
ℓ_4	0	0	1	0	1	1	0	0	1	1	1	1	1	1	0
ℓ_5	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
ℓ_6	0	0	0	0	0	0	1	0	0	0	1	0	1	1	1

Figure 8.5: Example of a solution for a set partitioning problem, the rows are associated with legs, the columns with shifts. Shown is a solution of cost 10.

Fig. 8.5 shows an example of sub-instance with 6 legs $\ell_1, \ldots, \ell_6$ and 15 shifts $s_1, \ldots s_{15}$. Values 1/0 in the matrix indicate whether the leg at the row is covered by the shift at the column. Shifts have costs $c_{s_1}, \ldots c_{s_{15}}$, displayed in the first row. A solution $S = \{s_4, s_8, s_{14}\}$ to the set partitioning with cost 10 is shown.

Differently from other versions of CMSA, we do not set a time limit for CPLEX, which we always allow running until the optimal solution for the sub-instance is found. In general, it might be a bad idea not to set any time limit for the MIP solver, but the reason behind this choice in this context is that CPLEX is very efficient in solving the set partitioning problem when the size of the sub-instance is kept reasonable. So, there are no real advantages in accepting sub-optimal or bad solutions from CPLEX, which may lead to the discard of good solution components in the adapt phase. Thus, besides the size of the specific tackled problem instance, the time and memory resources that CPLEX needs, on average, in the solve phase, depend solely on the values chosen for the three parameters n_{sols}, d_{rate}, and age_{limit}, which influence it as follows. When the parameter n_{sols} is set to high values, the sub-instance grows faster after each iteration, because more solution components are added to $\mathcal{C}'$. Lower values of d_{rate} imply that more diverse solution components are discovered by the greedy heuristic and added to the sub-instance, while high values generate more homogeneous shifts, many of which will be discarded due to being already present in the sub-instance. Finally, age_{limit} affects the memory of the algorithm: lower values imply that non-profitable solution components are discarded earlier and the size

of the sub-instance is kept smaller, while high values retain those solutions components longer in the sub-instance. For the above-mentioned reasons, finding a good balance of the parameters values through a proper tuning procedure is a crucial task, that is discussed in Section 8.4.1.

8.4 Experimental results

We tested the CMSA algorithm on the wide set of realistic instances available in the literature. Instances sizes range from 10 tours (about 70 legs) to 250 tours (at most 2500 legs). The instances with sizes 10–100 were released in Kletzander and Musliu (2020), while the larger instances with sizes 150–250 were introduced later in Kletzander et al. (2022).

We compare our CMSA with the state-of-the-art algorithms previously presented in the literature: Simulated Annealing (SA) and Hill Climbing (HC) (Kletzander and Musliu, 2020), Tabu Search (TS) (Kletzander et al., 2022), and three hyper-heuristics using low-level heuristics proposed in Kletzander and Musliu (2022): Chuang-Pruning (CH-PR) (Chuang, 2020), a combination of adaptive mechanisms to manage a set of active low-level heuristics (GIHH) (Misir et al., 2011), and a streamlined (lean) version of GIHH (L-GIHH) (Adriaensen and Nowé, 2016). We compare the results also with the Branch and Price (B&P) developed by Kletzander et al. (2021).

We implemented the CMSA in C++ and compiled with GNU g++, version 9.4.0 in `-O3` mode, on Ubuntu 20.04.4 LTS. The experiments were run on a machine equipped with an AMD Ryzen Threadripper PRO 3975WX processor with 32 cores, with a base clock frequency of 3.5 GHz, 64 GB of RAM. We allowed one core per experiment. The experiments for other algorithms were run on a different and slower machine, with a base clock frequency of 2.20 GHz and max frequency of 2.90GHz.

Although a completely fair comparison is not possible, for the above-mentioned reasons and because the algorithms were not all implemented in the same programming language, experimental data presented in Section 8.4.2 clearly shows that CMSA is able to outperform other metaheuristics on most instance classes, even if the time limit for CMSA is kept much shorter than for other methods.

8.4.1 Parameter tuning

We tuned the values for parameters n_{sols}, d_{rate} and c_{list} through the automatic algorithm configuration tool JSON2RUN, described in Chapter 4. We independently tuned the parameters for the instances with sizes from

Table 8.2: CMSA parameters, the considered domains for parameter tuning, and the finally determined parameter values.

	10-100 tours		150-250 tours	
Parameter	Domain	Value	Domain	Value
n_{sols}	$\{2, 3, ..., 500\}$	300	$\{2, 3, ..., 200\}$	66
d_{rate}	$[0.50, 1.00]$	0.77	$[0.80, 1.00]$	0.96
age_{limit}	$\{2, 3, ..., 50\}$	4	$\{2, 3, ..., 30\}$	4

Table 8.3: Average results (costs) for classes of instances (sizes expressed by number of tours) and methods.

Size	CMSA	SA	HC	TS	CH-PR	GIHH	L-GIHH	B&P
10	14879,7	**14739,6**	14988,4	15036,4	14956,2	14847,4	14810,6	14709,2
20	**30745,9**	30971,0	31275,6	31248,4	30896,7	30892,2	30810,8	30294,8
30	**50817,2**	51258,0	51917,4	51483,0	51331,4	51059,4	51037,6	49846,4
40	**68499,9**	69379,8	71337,6	69941,2	69182,9	68988,4	69022,2	67000,4
50	**86389,2**	87557,4	87262,4	87850,6	87394,3	87184,4	87145,2	84341,0
60	**102822,9**	104333,0	104296,4	104926,2	103921,5	103491,6	103467,3	99727,0
70	**121141,9**	123225,6	123304,0	123632,2	122502,9	122198,6	122321,8	118524,2
80	**138760,3**	140914,0	140508,0	140482,4	139931,8	139648,2	139551,9	134513,8
90	**155078,3**	157426,0	156862,4	156296,4	155520,8	155560,8	155649,6	150370,8
100	**171786,7**	174501,8	172909,0	172916,0	171901,0	171879,8	172763,7	172582,2
150	**263387,7**	266705,5	265492,3	265654,8	-	-	-	-
200	**349017,0**	354408,4	353494,9	350747,2	-	-	-	-
250	**439234,5**	446525,0	446000,9	443845,8	-	-	-	-

10 to 100 tours and for the new larger instances, with sizes spread from 150 to 250 tours. Indeed, we had to allow smaller domains for the larger instances because combinations of high values of age_{limit} and n_{sols} together with small d_{rate} are very likely to give birth to ILP models that are too large and that may saturate the memory during the solve phase. Parameters n_{sols} and age_{limit} have domains of natural numbers, while d_{rate} takes real numbers with a precision of two decimal places. Table 8.2 shows the domains that we applied to the parameters and the different outcomes of the tuning procedures.

Table 8.4: CMSA results (costs) measured after 15, 30, and 60 minutes (900, 1800 and 3600 s), and comparison with state-of-the-art metaheuristics and B&P. Best values among metaheuristic methods are in bold.

Instance	CMSA - average			Benchmark		B&P	
size	900s	1800s	3600s	method	3600s	time	best
10	14899,0	14886,4	14879,7	SA	**14739,6**	7,2	14709,2
20	**30805,1**	**30770,3**	**30745,9**	L-GIHH	30810,8	1201,4	30294,8
30	**50911,6**	**50863,0**	**50817,2**	L-GIHH	51037,6	3610,6	49846,4
40	**68711,3**	**68600,2**	**68499,9**	GIHH	68988,4	3605,8	67000,4
50	**86674,3**	**86517,0**	**86389,2**	L-GIHH	87145,2	3674,4	84341,0
60	**103206,0**	**102998,3**	**102822,9**	L-GIHH	103467,3	4373,2	99727,0
70	**121734,6**	**121410,7**	**121141,9**	GIHH	122198,6	6460,4	118524,2
80	**139397,4**	**139073,7**	**138760,3**	L-GIHH	139551,9	5912,4	134513,8
90	155674,5	**155387,4**	**155078,3**	CH-PR	155520,8	7390,4	150370,8
100	172447,3	172086,9	**171786,7**	L-GIHH	171833,5	7395,8	172582,2
150	**264261,6**	**263803,9**	**263387,7**	HC	265492,3	-	-
200	**350638,9**	**349707,2**	**349017,0**	TS	350747,2	-	-
250	**441917,3**	**440364,5**	**439234,5**	TS	443845,8	-	-

8.4.2 Analysis of the results

Table 8.3 shows the average results grouped by instance sizes for different methods. Each instance size class contains five distinct instances and we executed 10 independent runs on each instance, so that each value is calculated over 50 runs. The values for SA and HC are also taken over 10 runs per instance, while for the hyper-heuristics 5 runs per instance were executed. TS and B&P are deterministic, so runs are not repeated. All algorithms worked with time limits of 1 hour, except for B&P, which was allowed up to 2 hours. Values in bold report best results within metaheuristics, while underlined values are the best values including also the exact approach. We can observe that CMSA outperforms other metaheuristics on all instance groups but the smallest one, sized 10. In general, the best results for instances up to size 90 remain the one set by the B&P, whilst, for larger instances, CMSA gets the new best results. For larger instances sized 150, 200 and 250 tours, only data for SA, HC and TS are available for comparison.

Table 8.4 shows mean values of the objective function collected from the same CMSA experiments as those presented in Table 8.3 after 15 minutes (900 s) and 30 minutes (1800 s). We compare them with the state-of-the-art metaheuristic, which is specified in the column *benchmark*. We report also the results of the B&P, which has a time limit of 2 hours, but it may stop

before, if an optimal solution is found, so actual B&P execution time is specified as well. Results that improve or equal the current state-of-the-art within metaheuristics are marked in bold.
Data show that CMSA converges very quickly toward good solutions. After 15 minutes it already shows better results than other metaheuristics for 10 out of 13 instance classes, and for 11 out of 13 after 30 minutes. For instances sized 100, CMSA after 15 or 30 minutes is already capable to perform better also than the exact method in 2 hours, but not better than the hyper-heuristic L-GIHH. Data suggest also that CMSA is not likely to get stuck on early local minima, as we can see always a consistent decrease of the cost function value over time.
Finally, the fact that CMSA is able to provide good solutions quickly may be interesting for real-world applications, where human decision makers are likely to prefer to wait short times to have in hand the results of the automated scheduling.

8.5 Conclusions

We applied the CMSA metaheuristic to BDS, a complex and challenging real-world problem that integrates scheduling and routing issues. CMSA turned out to compare favorably with the state-of-the-art metaheuristics for this problem. In particular, it showed good performances on the large instances, which are in general the most critical ones.

Chapter 9

Maximum Disjoint Dominating Sets Problem

Let $G = (V, E)$ be an undirected graph where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. A *dominating set* in G is a set of vertices $\mathcal{D} \subseteq V$ such that every vertex $v \in V \setminus \mathcal{D}$ is adjacent to at least one vertex $v' \in \mathcal{D}$. The decision problem of determining whether a dominating set of size $|\mathcal{D}| \leq K$ exists is NP-complete (Garey and Johnson, 1979), whereas the related Minimum Dominating Set problem, which requires finding the smallest dominating set in a given graph, is NP-hard (Irving, 1991). In this work, we are interested in the Maximum Disjoint Dominating Sets Problem, in which a valid solution $\mathcal{S} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ consists of a collection of disjoint dominating sets $\mathcal{D}_i$ $(i = 1, \ldots, k)$ of G, where disjoint means that $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ for all $i \neq j \in \{1, \ldots, k\}$. The objective function value $f(\mathcal{S})$ of a valid solution $\mathcal{S}$ is the number of disjoint dominating sets in $\mathcal{S}$, that is, $f(\mathcal{S}) := |\mathcal{S}|$. The goal is to find a valid solution $\mathcal{S}^*$ that maximises f. This description applies to the DPP as well, with the supplementary condition that $\bigcup_{\mathcal{D} \in \mathcal{S}} \mathcal{D} = V$.

In addition to its research significance from a theoretical and computational point of view, solving the MDDSP is relevant from a real-world application perspective. Its utility is found, for example, in heterogeneous multi-agent systems (Mesbahi and Egerstedt, 2010) and in wireless sensor networks (WSN). In particular, in WSNs disjoint dominating sets find applications in mechanisms for sleep-wake cycling (Cardei et al., 2002; Cardei and Du, 2005). The aim of such a mechanism is to prolong the lifetime of a battery-powered sensor network. In fact, the expected lifetime of a sensor network equipped with a disjoint-sets-based sleep-wake cycling

mechanism is directly proportional to the number of disjoint dominating sets that can be found in the graph.

Note that finding at least one dominating set in a graph is always possible and trivial, given that the node set V of the graph is a dominating set of $G = (V, E)$. However, note that a solution $\mathcal{S} = \{V\}$, which has an objective function value of $f(\mathcal{S}) = |\mathcal{S}| = 1$, is the worst solution that can be found, even though it exists in each input graph. Furthermore, every graph without isolated vertices contains at least two disjoint dominating sets (Ore, 1962). In general, the number of dominating sets on graphs is a number between 1 and $\delta(G) + 1$, where $\delta(G)$ is the minimum degree of all vertices in graph G. Indeed, $\delta(G) + 1$ is a proven upper bound for the number of disjoint dominating sets in G (Cockayne and Hedetniemi, 1977). It is actually rather easy to verify this upper bound. Given a solution $\mathcal{S} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ for the input graph G, any vertex $v \in V$ must be dominated by a different vertex in all $\mathcal{D}_i \in \mathcal{S}$. However, a vertex $v \in V$ with degree $deg(v)$ can only be dominated by itself or by any of its neighbors, that is, by at most $deg(v)+1$ different vertices. Therefore, there will be no more than $\delta(G) + 1$ disjoint dominating sets in G. Nevertheless, this value, which is very easy to calculate and definitely useful when solving the problem in practice, does not imply that an actual solution with $\delta(G) + 1$ disjoint dominating sets exists. That is to say, a solution with $\delta(G) + 1$ dominating sets is surely optimal, but there is no guarantee that the given input graph contains a solution of value $\delta(G) + 1$.

Finally, we introduce some general concepts on undirected graphs that are required later in this chapter. The open neighborhood of vertex v is $N(v) := \{u \in V \mid (u, v) \in E\}$, which represents the set of vertices adjacent to v in G. The closed neighborhood of vertex v is $N[v] := N(v) \cup \{v\}$, that is, $N[v]$ includes all vertices adjacent to v and v itself. The number of neighbors of v corresponds to its degree, which is denoted by $deg(v)$. In other words, $deg(v) = |N(v)|$.

Note that a highly related optimization problem is the Domatic Partition Problem. Given a simple, undirected graph $G = (V, E)$ the DPP problem requires to partition the set of vertices V into the maximal number of disjoint dominating sets. In other words, a valid solution $\mathcal{S} = \{D_1, \ldots, D_k\}$ to the DPP problem not only requires that all pairs of dominating sets $D_i \neq D_j \in \mathcal{S}$ are disjoint $(i, j = 1, \ldots, k)$, but also that $\bigcup_{i=1}^{k} D_i = V$. Nevertheless, observe that any solution $\mathcal{S}$ to the MDDSP can easily be transformed into a solution $\mathcal{S}'$ to the DPP by adding all vertices from $V \setminus \bigcup_{i=1}^{|\mathcal{S}|} D_i$ to any of the disjoint dominating sets of $\mathcal{S}$. Note that, by adding further

vertices to a dominating set D, set D does not lose its property of being a dominating set. This implies that an optimal solution to the MDDSP can easily be transformed into an optimal solution to the DPP. The value of an optimal solution to the DPP in graph G, denoted by $\gamma(G)$, is also called the *domatic number* of G. In this context, the term "domatic" was created as a composition of "dominating" and "chromatic" (Chang, 1994). In the related literature, one can also find numerous references to the so-called Domatic Number Problem (DNP) (Chang, 1994). However, this problem only refers to finding the domatic number $\gamma(G)$ of a graph G. In other words, algorithmic approaches for solving the DNP try to identify $\gamma(G)$ without necessarily generating a corresponding solution. Therefore, by solving the DPP we simultaneously solve the DNP, but not vice versa. Finally, a special case of the DPP is the k-domatic partition problem that seeks to find a partition of a given graph into k disjoint dominating sets. Correspondingly, the k-domatic number problem asks whether a given graph can be partitioned into k dominating sets.

9.1 Graphical problem illustration

Consider the undirected graph $G = (V, E)$, with 11 vertices ($|V| = 11$) and 17 edges ($|E| = 17$), displayed in Fig. 9.1. Vertices are labelled $v_1, \ldots, v_{11}$, while edges are unlabelled. An optimal solution is $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$, where $\mathcal{D}_1 = \{v_1, v_4, v_6, v_8\}$, $\mathcal{D}_2 = \{v_2, v_5, v_{10}\}$, and $\mathcal{D}_3 = \{v_3, v_7, v_9\}$. The three sets are represented in the figure with three different background colors. It is easy to verify that all these sets are dominating sets. We show it for $\mathcal{D}_1$:

1. v_1 dominates the adjacent vertices v_2 and v_3
2. v_4 dominates v_2, v_3 and v_5
3. v_6 dominates v_5, v_{10} and v_9
4. v_8 dominates v_7, v_{10} and v_{11}

The same holds for $\mathcal{D}_2$ and $\mathcal{D}_3$. Furthermore the three sets are disjoint because $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$, $\mathcal{D}_1 \cap \mathcal{D}_3 = \emptyset$, and $\mathcal{D}_2 \cap \mathcal{D}_3 = \emptyset$. Hence, $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ is a solution to the MDDSP in graph G, with objective function value $f(\mathcal{S}) = 3$. If we add v_{11} to any of the dominating sets, $\mathcal{S}$ becomes a partition of V and, so, a valid and optimal solution also for the DPP.

Finally, we show through an example that an optimal solution with objective value $\delta(G) + 1$, which we discussed above, does not exist in all

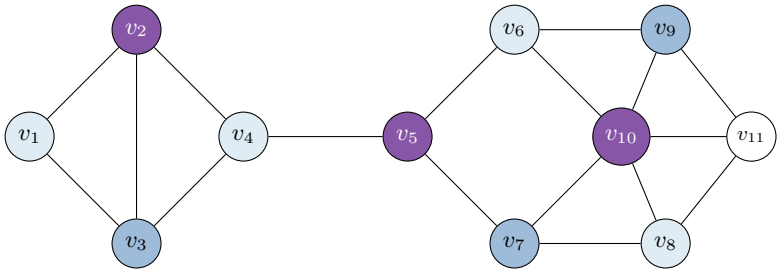


Figure 9.1: A graph with 11 vertices and 17 edges. The upper bound for the domatic number is $\delta(G) + 1 = 3$. Moreover, there exists an optimal solution with 3 disjoint dominating sets (as indicated by the vertices with a background color different to white).

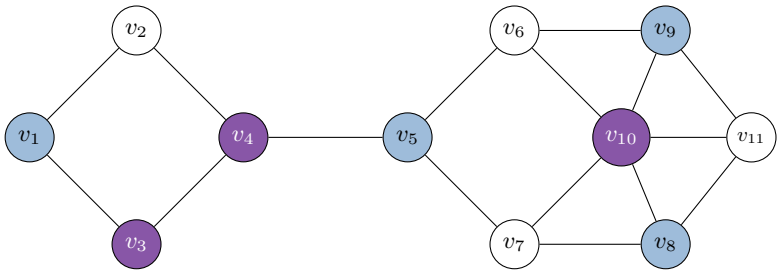


Figure 9.2: A graph with 11 vertices and 16 edges. The upper bound for the domatic number is $\delta(G)+1 = 3$, but an optimal solution has only 2 disjoint dominating sets. One of the possible optimal solutions is indicated by blue and purple vertices. Uncolored vertices do not belong to any dominating set of the displayed solution.

graphs. Fig. 9.2 represents the same graph of Fig. 9.1, without the edge that connects vertices v_2 and v_3. In both examples, $\delta(G) + 1 = 2 + 1 = 3$. We have seen above that a solution with value $f(\mathcal{S}) = 3$ exists for the graph in Fig. 9.1. However, a solution $\mathcal{S}' = \{\mathcal{D}_4, \mathcal{D}_5\}$, where $\mathcal{D}_4 = \{v_1, v_5, v_8, v_9\}$, and $\mathcal{D}_5 = \{v_3, v_4, v_{10}\}$, with a value of $f(\mathcal{S}') = 2$, is already an optimal solution for the graph in Fig. 9.2, and there are no solutions with three dominating sets. Solution $\mathcal{S}'$ is represented in Fig. 9.2 by means of different background colors for the vertices.

9.2 Related work

The identification of small dominating sets in graphs and networks is one of the classical combinatorial optimization problems in graph theory, with numerous applications ranging from biology to communication networks (Haynes et al., 2013). Interest in the MDDSP and related problems dates back to the 70s. The first mention of the Domatic Number Problem, which deals with deriving the domatic number of a given graph without the need for deriving the corresponding disjoint dominating sets, is found in the work of Cockayne and Hedetniemi (1975). Many other studies are found in the literature on the DNP. Most of them are primarily interested in specific families of graphs. Although not many exact approaches can be found in the literature for MDDSP-like problems, mainly due to their complexity, some exact approaches have been designed for finding optimal solutions to the DNP. In particular, the first exact deterministic exponential-time algorithm for the 3-DNP was designed by Riege and Rothe (2005). The algorithm has a running time of $\mathcal{O}(2.9416^n)$ and uses polynomial space, which is in contrast to a naive approach that runs in $\mathcal{O}(3^n)$ of time. This time complexity was later improved to $\mathcal{O}(2.695^n)$ by Riege et al. (2007). Combining the two main techniques typically used for the design of exact exponential-time algorithms–inclusion & exclusion, respectively measure & conquer– Van-Rooij (2010) provided a fast polynomial-space algorithm that computes the domatic number in $\mathcal{O}(2.7139^n)$ time.

The Domatic Partition Problem was first introduced by Cockayne and Hedetniemi (1977), two years after the DNP. In the same paper, the authors showed that the problem has an upper bound of $\delta(G)+1$, where $\delta(G)$ is the minimum degree of all vertices in G. In other words, an optimal solution to the DPP can never have more disjoint dominating sets than $\delta(G)+1$. Furthermore, the problem was shown to belong to the class of NP-hard problems on general graphs (Garey and Johnson, 1979). In fact, it remains NP-hard for co-bipartite graphs (Poon et al., 2012). Moreover, unless P=NP, the MDDSP has no polynomial-time α-approximation algorithm for any constant α smaller than 1.5 (Cardei et al., 2002). Existing polynomial-time approximation schemes can be found in Feige et al. (2002). In addition to the above, the NP-completeness of the 3-domatic partition problem was proofed for general graphs by Cardei et al. (2002) and still holds when restricted to planar bipartite graphs (Poon et al., 2012) and planar unit disk graphs (Nguyen and Huynh, 2007).

Practical applications of this problem can be found especially in the context of heterogeneous multi-agent systems (Mesbahi and Egerstedt,

2010) and in wireless sensor networks (WSN), which are studied for their applications in the fields of healthcare, environmental monitoring, emergency operations and security surveillance (Akyildiz et al., 2002). Note that WSNs are networks composed of a rather large number of small devices called sensor nodes. These sensor nodes capture information from the environment and they are also responsible for transmitting the captured data to a base station. The power supply of sensor nodes is generally provided through batteries, which implies that their lifetime is limited. For the purpose of energy conservation, sensor nodes may change from their normal, active mode to another mode called low-energy, respectively sleep, mode. Hereby, the active mode allows capturing information and transmitting data, for example. The difference of energy consumption between the sleep and the active mode is considerable and may reach about two orders of magnitude (Cardei et al., 2002; Cardei and Du, 2005). When grouped into disjoint dominating sets, at any moment in time only the sensor nodes belonging to exactly one of these dominating sets are in active mode. All the others remain in sleep mode. Moreover, when the currently active sensor nodes reach a critical battery level they are put into sleep mode and the sensor nodes of the next dominating set are activated. This is repeated until all disjoint dominating sets have been used. Thus, the expected lifetime of the network is determined by the number of disjoint dominating sets times the average lifetime of a sensor in the active state. Therefore, the higher the number of dominating sets, the longer is the lifetime of the WSN.

A natural way to produce a feasible solution to the MDDSP is to greedily construct dominating sets of preferably small cardinality with the ultimate goal to maximize the number of disjoint dominating sets that can be generated.[1] In this context, many greedy heuristic strategies have been proposed in the literature and experimentally tested, such as the ones given in Cardei et al. (2002) (Color-Dds), Nguyen and Huynh (2007) (P-Max, P-Min, R-Lid), Islam et al. (2009) (Iam) ,Balbal et al. (2021) (Mdds-Gh). They are discussed in detail in Section 9.4.1. Additionally, Landete and Sainz-Pardo (2022) presented an exact decomposition algorithm for finding a domatic partition on separable graphs, that is, graphs with at least one node (cut-vertex) whose removal produces two or more connected components.

[1] Remember that any solution to the MDDSP can be trivially transformed to a solution to the DPP by adding those vertices that do not belong to any of the disjoint dominating sets to one of the dominating sets of the MDDSP-solution.

9.3 Integer linear programming formulations

A natural ILP model for the MDDSP on graph $G = (V, E)$ is based on binary variables $x_{vs} \in \{0, 1\}$ for every vertex $v \in V$ and every theoretically possible dominating set $s = 1, \ldots, \delta(G) + 1$. If v is in the s-th dominating set, then $x_{vs} = 1$, zero otherwise. A second binary variable $y_s \in \{0, 1\}$ tells whether the s-th dominating set is chosen to be part of the solution. Given these two sets of variables, the ILP model for MDDSP reads as follows.

$$\max \sum_{s=1}^{\delta(G)+1} y_s \tag{9.1a}$$

$$\sum_{s=1}^{\delta(G)+1} x_{vs} \leq 1 \qquad \forall v \in V \tag{9.1b}$$

$$\sum_{u \in N(v)} x_{us} \geq y_s - x_{vs} \qquad \forall v \in V, \forall s \in \{1, \ldots, \delta(G) + 1\} \tag{9.1c}$$

$$y_s \geq x_{vs} \qquad \forall v \in V, \forall s \in \{1, \ldots, \delta(G) + 1\} \tag{9.1d}$$

$$x_{vs} \in \{0, 1\}, y_s \in \{0, 1\} \qquad \forall v \in V, \forall s \in \{1, \ldots, \delta(G) + 1\} \tag{9.1e}$$

The objective at Eq. (9.1a) maximises the number of dominating sets in the solution. Constraints at Eq. (9.1b) ensure that the dominating sets are disjoint, requiring each vertex v to be assigned to at most one dominating set. Constraints at Eq. (9.1c) ensure that the sets that are chosen in the solution are dominating sets. To evaluate this constraint, we make use of the concept of open neighborhood. Constraints at Eq. (9.1d) ensure that nodes are only assigned to active dominating sets. Finally, Constraints at Eq. (9.1e) define the binary nature of the variables.

This model, however, does not include any symmetry breaking constraints. Note that symmetric–and, therefore, redundant–solutions are obtained simply by permuting the sets of a solution. In that way, a solution is obtained with exactly the same sets, just that the set indices/names are different. In particular, given a solution $\mathcal{S}$ with $|\mathcal{S}|$ sets, symmetric solutions are obtained by all possible $|\mathcal{S}|$-permutations of the $\delta(G)+1$ indices, that is, every solution $\mathcal{S}$ can be expressed in $\frac{(\delta(G)+1)!}{(\delta(G)+1-|\mathcal{S}|)!}$ ways. Adding symmetry breaking constraints to the ILP model reduces significantly the size of the search space, at the expense of a higher number of constraints that lead to an increased complexity.

In the following, we present two additional ILP models that implement two different symmetry breaking strategies. Both are obtained by adaptations of the constraints proposed by Méndez-Díaz and Zabala (2008) for graph coloring. For sake of clarity, we name the model described above as ILP, and the two models with symmetry breaking constraints as ILP-SM$_1$ and ILP-SM$_2$.

9.3.1 Model 1 with symmetry breaking constraints

Model ILP-SM$_1$ imposes that the number of vertices assigned to the s-th set must be greater or equal than the number of vertices in the $s+1$-th set. This is done by adding to Model ILP the following inequalities:

$$\sum_{v \in V} x_{v,s} \geq \sum_{v \in V} x_{v,s+1} \qquad \forall s = 1, ..., \delta(G) \tag{9.2}$$

Constraints in Eq. (9.2) lead to an ordering of the sets by decreasing size, that is, set s must represent a set with a higher or equal cardinality than the s-th set.

9.3.2 Model 2 with symmetry breaking constraints

Méndez-Díaz and Zabala point out in their work that Model ILP-SM$_1$ does not prevent symmetries that originate from permutations of the indices within sets of the same size, a scenario that is likely to arise in the MDDSP. This is solved in Model ILP-SM$_2$ by adding to Model ILP the following constraints:

$$x_{v_i,s} = 0 \qquad \forall s \geq i + 1 \tag{9.3a}$$

$$x_{v_i,s} \leq \sum_{k=s-1}^{i-1} x_{v_k,s-1} \qquad \forall i \in V \setminus \{1\}, \forall 2 \leq s \leq i - 1 \tag{9.3b}$$

Model ILP-SM$_2$ assigns to sets of vertices a set index that corresponds to the smallest index of all vertices in the set. This model, therefore, also breaks the symmetries that originate from permutations of the indices within sets of the same size.

9.4 Multi-Constructor CMSA for the MDDSP

In this work, we propose the Multi-Constructor CMSA, described also in Chapter 3. In the construction step of the Multi-Constructor CMSA, every

solution is generated by a constructor chosen at random from a pre-defined portfolio. Hereby, each constructor has a weight value assigned, and the probability to be chosen depends on these weight values. In particular, we consider all six greedy heuristics from Section 9.4.1 as constructors for our CMSA approach. In order to produce (possibly) different solutions at each call to a constructor, they are used in a probabilistic way, which work as follows: at every greedy step, the classical deterministic greedy function is applied with a probability equal to the value of the parameter $0 \leq d_{\text{rate}} < 1$.[2]. Otherwise, a candidate list containing the first c_{list} candidates, sorted by value of their greedy function from best to worst, is built, and a solution component is drawn uniformly at random from the list.

Our choice of CMSA for tackling the MDDSP is motivated by the fact that this metaheuristic has already shown a high potential for related optimization problems in graphs. Examples include the minimum capacitated dominating set problem (Pinacho-Davidson et al., 2019) and the maximum happy vertices problem (Lewis et al., 2019). Furthermore, CMSA has been shown to perform well for real-world problems such as the prioritized pairwise test data generation problem (Ferrer et al., 2021) or the bus driver scheduling problem with complex break constraints (Chapter 8).

Hereafter, we discuss the main components of our Multi-Constructor CMSA, namely, the *constructors*, the *sub-instance*, the *model* solved in the SOLVE phase and the *lexicographic objective* function used internally by the metaheuristic.

9.4.1 Constructors

Given the existence of effective greedy heuristics from the literature for the MDDSP, mentioned in Section 9.2, we adapt them as constructor in CMSA. In particular, we take as constructors for our CMSA approach the following six greedy heuristics: COLOR-DDS, P-MAX, P-MIN, R-LID, IAM, MDDS-GH. We suitably randomize them to produce different solutions at each call, based on two parameters: $d_{rate} \in [0,1]$ and $c_{list} \in \mathbb{N}$. We denote the randomized version of a greedy with a letter r. The resulting multi-constructor is:

$$\mathcal{H} = \text{MDDS-GH}_r \cup \text{IAM}_r \cup \text{P-MAX}_r \cup \text{P-MIN}_r \cup \text{R-LID}_r \cup \text{COLOR-DDS}_r \quad (9.4)$$

We provide in this section a rather detailed description of these heuristics and the way we randomized them.

[2] d_{rate} stands for "determinism rate"

Color-Dds$_r$

The first greedy heuristic for the MDDSP was provided by Cardei et al. (2002). It is a vertex-coloring heuristic (henceforth called Color-Dds) working in two phases and with a time complexity of $O(|V|^3)$. In the first phase, all vertices are colored using the sequential Welsh-Powell coloring algorithm (Welsh and Powell, 1967) to generate all possible independent dominating sets. Each independent set is formed by vertices with the same color, where colors are indicated by numbers. Remember that an independent set in a graph is a subset of vertices such that no two vertices in it are adjacent. In the second phase, for each independent set in ascending order of the color identifiers, the algorithm checks whether it represents a dominating set or not. More specifically, if (1) the current independent set is not a dominating set and (2) there is no possibility to achieve that by adding some vertices from other independent sets with color greater than the current one, then the termination condition of the algorithm is met. A pseudo-code for the Color-Dds heuristic is shown in Fig. 9.3.

procedure Color-Dds(an undirected graph $G = (V, E)$)
1: Let $\{v_1, v_2, \cdots, v_n\}$ be a sorting of $v \in V$ in descending order of degree
2: color$[v_1] \leftarrow 1$
3: **for** i in $2, \ldots, n$
4: color v_i with the smallest possible color (greater or equal to 1) not appearing in any neighbor v_j of v_i with $j < i$.
5: $n_{\text{colors}} \leftarrow \max\{\text{color}[v] \mid v \in V\}$
6: $\mathcal{D}_1 \leftarrow \{v \in V \mid \text{color}[v] = 1\}$
7: $k \leftarrow 1$
8: **for** k in $1, \ldots, \min\{\delta(G) + 1, n_{\text{colors}}\}$
9: $\mathcal{D}_{k+1} \leftarrow \{v \in V \mid \text{color}[v] = k + 1\}$
10: **for** all $v \in V$ s.t. color$[v] < k + 1$
11: **if** v is not dominated by a vertex with color $k + 1$
12: **if** v has a neighbor u with largest color greater than $k + 1$
13: color$[u] \leftarrow k + 1$
14: $\mathcal{D}_{k+1} \leftarrow \mathcal{D}_{k+1} \cup \{u\}$
15 **else**
16: **return** (go to line 17)
17: **return** $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k\}$

Figure 9.3: Color-Dds greedy heuristic for the MDDSP

In order to provide a randomized version Color-Dds$_r$ of the algorithm, we modify line 1 of the algorithm as follows: when sorting, with probability

d_{rate}, we insert the next remaining vertex with higher degree, while with probability $1 - d_{rate}$ we select among the next c_{list} vertices with highest degree, or among all the remaining vertices, if less than c_{list}.

P-MAX$_r$, P-MIN$_r$ and R-LID$_r$

Nguyen and Huynh (2007) proposed three other deterministic greedy heuristics, namely Progressive Maximum Degree Disjoint Dominating Sets (P-MAX), Progressive Minimum Degree Disjoint Dominating Sets (P-MIN) and Random Lowest ID Disjoint Dominating Sets (R-LID). These heuristics are two-step processes and adopt a similar mechanism in which a collection of disjoint dominating sets $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_{|\mathcal{S}|}\}$ is formed by successively constructing $\mathcal{D}_k$, starting from an empty set for all $k = 1, \ldots, |\mathcal{S}|$. The construction of a dominating set $\mathcal{D}_k$ at step k for each of these greedy heuristics is done as follows. First, $\mathcal{D}_k$ is initialized to the empty set. Then, at each construction step, vertices of the input graph $G(V, E)$ are classified into three distinct sets with respect to $\bigcup_{i=1}^{k} \mathcal{D}_i$:

(i) `BLACK` vertices: vertices contained in $\bigcup_{i=1}^{k} \mathcal{D}_i$, that is, vertices inserted in one of the already generated dominating sets (including the current partial dominating set).

(ii) `GREY` vertices: vertices that are not `BLACK` but adjacent to some `BLACK` vertex.

(iii) `WHITE` vertices: all vertices from V that are neither `BLACK` nor `GREY`.

Given this classification, only `WHITE` vertices can be added to the current partial dominating set. In order to be able to make a choice, each `WHITE` vertex v is first evaluated by the following greedy function:

$$score^P(v) := \left| \left\{ u \in N(v) \mid u \text{ is a } \texttt{WHITE} \text{ vertex with respect to } \bigcup_{i=1}^{k} \mathcal{D}_i \right\} \right| \tag{9.5}$$

The three greedy heuristics from Nguyen and Huynh differ then in how a `WHITE` vertex is chosen at each construction step. In the case of P-MIN and P-MAX the next `white` vertex to be placed in $\mathcal{D}_k$ is the one with the maximum, respectively minimum, value of the greedy function $score^P()$. R-LID, on the other side, does not make use of this greedy function. It simply prefers the `WHITE` vertex with the lowest index. When no further `WHITE` vertex can be added to $\mathcal{D}_k$, $\mathcal{D}_k$ may still not be a valid dominating

set. Therefore, uncovered vertices are processed one after the other. In particular, such a vertex becomes covered if it has at least one `GREY` neighbor that can be added to $\mathcal{D}_k$. If it is not possible to cover all vertices, then the algorithm termination criterion is reached prior to the construction of the next dominating set $\mathcal{D}_{k+1}$. The pseudo-code of P-Max is presented in Fig. 9.4. Note that P-Min is obtained from Fig. 9.4 by replacing line 11 with the following instruction:

$$v^* \leftarrow \text{argmin}\{score^P(v) \mid v \in V\} \tag{9.6}$$

procedure P-Max(an undirected graph $G = (V, E)$)
1: color$[v] \leftarrow$ `WHITE` for all $v \in V$
2: $dominating\_set\_flag \leftarrow$ **true**; $k \leftarrow 1$
3: **while** $dominating\_set\_flag =$ **true**
4: **for** all $v \in V$
5: covered$[v] \leftarrow$ **false**
6: **if** (color$[v] =$ `GREY`) **then** color$[v] \leftarrow$ `WHITE`
7: $\mathcal{D}_k \leftarrow \emptyset$; $vertex\_cover\_flag \leftarrow$ **true**
8: **while** $vertex\_cover\_flag =$ **true**
9: **if** $\nexists v \in V$ **s.t.** color$[v] =$ `WHITE` **then** $vertex\_cover\_flag \leftarrow$ **false**
10: **else**
11: $v^* \leftarrow \text{argmax}\{score^P(v) \mid v \in V\}$
12: $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{v^*\}$
13: **for** all neighbors u of v^*
14: **if** (color$[u] =$ `WHITE`) **then** color$[u] \leftarrow$ `GREY`
15: covered$[u] \leftarrow$ **true**
16: color$[v^*] \leftarrow$ `BLACK`; covered$[v^*] \leftarrow$ **true**
17: **for** all vertex $v \in \bigcup_{i=1}^{k-1} \mathcal{D}_i$ **s.t.** covered$[v] =$ **false**
18: **if** v has a grey neighbor u **s.t.** $u \notin \bigcup_{i=1}^{k} \mathcal{D}_i$
19: $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{u\}$; color$[u] \leftarrow$ `BLACK`
20: **for** all neighbors w of u **do** covered$[w] \leftarrow$ **true**
21: **else** $dominating\_set\_flag \leftarrow$ **false**
22: **if** $dominating\_set\_flag$ **then** $k \leftarrow k + 1$
23: **return** $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_{k-1}\}$

Figure 9.4: Greedy heuristic P-Max for the MDDSP

In the case of P-Max$_r$, P-Min$_r$ and R-Lid$_r$, our randomized version modify line 11. With probability d_{rate}, we choose the next vertex according to the respective greedy rule of the algorithm (argmax, argmin, and lowest index, respectively). Otherwise, with probability $1 - d_{rate}$ we select among

the next c_{list} vertices with the highest score in the greedy function or among all the remaining vertices, if less than c_{list}.

IAM$_r$

Two years later, an improved version of P-MAX was introduced by Islam et al. (2009). This algorithm will henceforth be denoted as IAM, which is an acronym composed of the initials of the authors' surnames. In contrast to P-MAX, IAM is a one-step construction process of time complexity $O(n^3)$ in which the set of feasible solution components includes all vertices of V that are not part of the set of `BLACK` vertices as defined previously for the case of P-MAX. The employed greedy function is now defined as in Eq. (9.7) where ties are broken by choosing the vertex with the minimum number of `BLACK` neighbors. If the tie is still unresolved, the vertex with the lowest index is selected, similarly to what is done in R-LID

$$score^P(v) := |\{u \in N[v] \mid u \text{ is a } \texttt{WHITE} \text{ vertex with respect to } \mathcal{D}_k\}| \quad (9.7)$$

where $N[v]$ is the closed neighborhood of v, that is, the set of neighbors of v including itself.

Analogously to what we have done with the other greedy heuristics, we employ as constructor a suitably randomized version IAM$_r$. Again, with probability d_{rate}, we choose the next vertex according to Eq. (9.7), and with probability $1 - d_{rate}$ we select among the c_{list} vertices that score best in Eq. (9.7), or among the remaining vertices if less than c_{list}.

MDDS-GH$_r$

The last heuristic algorithm, and the currently best greedy algorithm, is called MDDS-GH. It was recently presented in Balbal et al. (2021). It was developed for a weighted variant of the MDDSP, and it was used as one of the principal components of a metaheuristic proposed for the same problem variant in Bouamama et al. (2022).

MDDS-GH can be easily adapted to tackle the MDDSP by incorporating IAM's greedy score function depicted in Eq. (9.7). This adaptation also improves the handling of algorithm termination criteria and enhances overall performance. It is worth noting that IAM faces an issue with one of its stopping conditions that can be met even when there is a possibility of forming additional remaining dominating sets. This aspect has been addressed and solved in MDDS-GH by using just one stopping condition. Unlike IAM, the algorithm checks, before beginning the current construction

process, if there is at least one vertex such that all its neighbors from the closed neighborhood are `BLACK` (part of previously constructed dominating sets). If this condition is met, no further dominating set can be found, and the algorithm terminates.

The algorithm MDDS-GH is comprehensively described in Balbal et al. (2021), therefore we refer to the pseudo-code shown in that article. In particular, despite the two greedy procedures being different, we designed its randomized version MDDS-GH$_r$ analogously to what we have done for IAM, given that both algorithms use the same greedy function. In reference to the pseudo-code shown in Algorithm 1 in the paper from Balbal et al., this affect line 12, in which the next candidate component is chosen.

Constructor selection

In our Multi-Constructor CMSA, the selection of the constructors is stochastic. Therefore, our constructors are assigned with a vector of probabilities $(\sigma_{\text{MDDS-GH}_r}, \sigma_{\text{IAM}_r}, \sigma_{\text{P-MAX}_r}, \sigma_{\text{P-MIN}_r}, \sigma_{\text{R-LID}_r}, \sigma_{\text{COLOR-DDS}_r})$, with $\sum_{i=1}^{K} \sigma_i = 1$, where σ_i is the probability of the i-th constructor. During the CONSTRUCT phase of CMSA, before the generation of a new solution, one of the constructors is selected by means of a biased random selection, according to the given probabilities. Then, the next solution is built using the selected constructor. A trivial choice would be to assign the same probability $1/K$ to all constructors. However, it is reasonable to believe that not all constructors are equally useful and that higher probabilities should be assigned to the most promising ones. Instead of using fixed rates for the constructor probabilities, we propose an online strategy for learning these probabilities, based on a *reinforcement learning* approach. Given that we dedicated the whole Chapter 10 to the use of reinforcement learning to adapt operator weights in metaheuristics, we forward the reader to that specific chapter for explanations on the methodology.

9.4.2 Sub-instance

The ILP model for the MDDSP from Section 9.3 uses certain constraints to ensure that the generated sets are both disjoint and dominating. These constraints are rather challenging, even for high-performance ILP solvers. Therefore, for our CMSA, we employ another paradigm based on the separation between (1) the generation of a large number of feasible dominating sets and (2) the subsequent selection of a collection of those sets such that the sets in the collection are vertex-disjoint. Theoretically, if we

had the means to enumerate the full collection $\mathcal{C}$ of all possible dominating sets of input graph $G = (V, E)$, an optimal solution to the MDDSP could be obtained by solving the following set packing ILP formulation:

$$\max \sum_{\mathcal{D} \in \mathcal{C}} x_{\mathcal{D}} \tag{9.8a}$$

$$\text{s.t.} \sum_{\{\mathcal{D} \in \mathcal{C} | v \in \mathcal{D}\}} x_{\mathcal{D}} \leq 1 \qquad \forall\, v \in V \tag{9.8b}$$

$$x_{\mathcal{D}} \in \{0, 1\} \qquad \forall\, \mathcal{D} \in \mathcal{C} \tag{9.8c}$$

This ILP model is based on a binary variable $x_{\mathcal{D}}$ for each dominating set $\mathcal{D} \in \mathcal{C}$, whereby a value of $x_{\mathcal{D}} = 1$ means that $\mathcal{D}$ is chosen to be part of the solution. Constraints at Eq. (9.8b) ensure that each vertex of G is present in at most one of the chosen dominating sets. In this way, the chosen dominating sets are pairwise disjoint.

However, in practice, there is no efficient way to enumerate all dominating sets of a reasonably large graph. And even if there was one, the size of $\mathcal{C}$ would be too large for the above ILP model to be solvable by nowadays' ILP solvers. On the contrary, we can efficiently generate a subset (or sub-instance) $\mathcal{C}' \subset \mathcal{C}$, containing a subset of all the possible dominating sets of G. The resulting ILP model where $\mathcal{C}'$ replaces $\mathcal{C}$ is tractable for a Mixed Integer Programming (MIP) solver, provided that the size of $\mathcal{C}'$ is kept reasonably small.

9.4.3 Lexicographic objective function

The MDDSP is characterized by the fact that many distinct solutions with identical objective function value coexist. This is because, generally, there are many different solutions with the same number of dominating sets. Indeed, given a solution $\mathcal{S} = \{D_1, \ldots, D_k\}$ to the MDDPS, the objective function value is simply $f(\mathcal{S}) := |\mathcal{S}|$, that is, it counts the number of disjoint dominating sets in $\mathcal{S}$. As a consequence, the search landscape is characterized by the presence of wide plateaus (Watson, 2010). The problem that arises in the presence of plateaus is that a metaheuristic, which is implicitly guided by gradients in the search landscape, may get lost. An effective way to deal with such a situation is to use a lexicographic objective function $f^{\text{lex}}()$, using more criteria than just the problem objective function in order to differentiate between different solutions. This is done, for example, in Bruglieri and Cordone (2021), where the original objective function is used as a first criterion for comparing two solutions, and, only when the two solutions have

the same original objective function value, a second criterion is evaluated to differentiate between them.

For the MDDSP, the idea to design the lexicographic objective function is that the most promising solutions on a plateau are those that are nearest to building an additional dominating set with the unused vertices. More precisely, given two solutions $\mathcal{S}_1$ and $\mathcal{S}_2$, $\mathcal{S}_1$ is said to be lexicographically better than $\mathcal{S}_2$–that is, $f^{lex}(\mathcal{S}_1) > f^{lex}(\mathcal{S}_2)$–if and only if

1. $f(\mathcal{S}_1) > f(\mathcal{S}_2)$ **or**
2. $f(\mathcal{S}_1) = f(\mathcal{S}_2)$ **and** $r(\mathcal{S}_1) > r(\mathcal{S}_2)$

Hereby, the second criterion, $r(\mathcal{S})$, is a residual coverage function that calculates the fraction of the input graph G that can be covered with the vertices $V' \subset V$ that, in a given a solution $\mathcal{S}$, are not assigned to any of the disjoint dominating sets. More specifically, $r(\mathcal{S})$ is defined as follows:

$$r(S) := \frac{\left|\bigcup_{v \in V'} N[v]\right|}{|V|} \tag{9.9}$$

In addition, the residual coverage function is used in CMSA (see lines 14–15 of Fig. 9.5) to discover if there are hidden dominating sets contained in the set of unused vertices regarding CPLEX solution $\mathcal{S}_{\text{exc}}$. Indeed, occasionally it may happen that $r(\mathcal{S}_{\text{exc}}) = 1$, which means that from the vertices not included in any dominating set of $\mathcal{S}_{\text{exc}}$ at least one additional dominating set can be generated. This may happen for two possible reasons: $\mathcal{C}'$ does not include such a dominating set because it was not generated by a constructor, or the time limit of t_{exc} CPU seconds did not allow the MIP solver to find an optimal solution to $\mathcal{C}'$. When this case is detected, the repair procedure is iteratively applied until $r(\mathcal{S}_{\text{exc}}) < 1$. The sets generated by the repair procedure are added to the sub-instance $\mathcal{C}'$ and their age is set to zero.

9.4.4 Parameters

Apart from graph G, our algorithm takes as input the values for seven parameters. Note that parameter values are determined through a statistically-principled tuning procedure, as discussed in Section 9.5.2. Five of these parameters, namely n_{sols}, d_{rate}, c_{list}, age_{limit}, and t_{exc}, are traditional CMSA parameters, while parameters λ and τ, which are needed for the reinforcement learning mechanism applied to the constructor probabilities,

Table 9.1: Parameters for CMSA

Param.	Explanation
n_{sols}	parameter that serves for the calculation of the number of solutions to be generated at each algorithm iteration, as explained below.
d_{rate}	determinism rate for solution construction.
c_{list}	length of the candidate list for those solution construction steps in which a non-deterministic choice is performed.
age_{limit}	limits the number of iterations a dominating set can remain in the sub-instance $\mathcal{C}'$ without being chosen by the exact solver for the best solution to the sub-instance.
t_{exc}	time limit (in seconds) for the application of the MIP solver at each iteration of CMSA.
λ	learning rate employed for the learning of constructor probabilities, only for the Multi-Constructor variant.
τ	minimum probability that can be reached by a constructor, only for the Multi-Constructor variant.

are specific to our Multi-Constructor CMSA. The full list of parameters is contained in Table 9.1.

We aim to find one single parameter value setting that works reasonably well for the whole range of problem instances. However, in preliminary experiments, we observed that the number of solution constructions allowed per iteration was very sensitive to the size of the input graph. In case of too many solution constructions, the resulting sub-instances have too many dominating sets, that is, the corresponding ILP models have too many variables, which makes it nearly impossible for CPLEX to find a good–or even optimal–solution within the fixed CPU time limit of t_{exc} seconds. This is because the number of vertices of the input graph ($|V|$) determines the number of rows of the corresponding ILP model, while the number of dominating sets in the sub-instance determines the number of columns. For this reason, we also observed that for smaller graphs many more solution constructions could be afforded when compared to larger graphs. Therefore, we finally decided to make the number of solution constructions allowed per algorithm iteration inversely proportional to $|V|$ by setting it to $\frac{n_{\text{sols}}}{|V|}$.

procedure MULTI-CONSTRUCTOR CMSA
(a graph $G(V,E)$, values for
parameters n_{sols}, d_{rate}, c_{list}, age_{limit}, t_{exc})
1: $t \leftarrow 1$
2: $(\sigma_{1,t}, \ldots, \sigma_{K,t}) \leftarrow \left(\frac{1}{K}, \ldots, \frac{1}{K}\right)$
3: $\mathcal{S}_{\text{bsf}} \leftarrow \emptyset$; $\mathcal{C}' \leftarrow \emptyset$
4: **while** $f(\mathcal{S}_{\text{bsf}}) < \delta(G) + 1$ **and** CPU time limit not reached
5: $\quad \hat{\mathcal{C}} \leftarrow \emptyset$
6: $\quad$ **for** $i \leftarrow 1, \ldots, \frac{n_{\text{sols}}}{|V|}$
7: $\quad\quad h \leftarrow \mathsf{ChooseConstructor}(\mathcal{H}, (\sigma_{1,t}, \ldots, \sigma_{K,t}))$
8: $\quad\quad \mathcal{S}_{\text{cur}} \leftarrow \mathsf{Construct}(G,h)$; $\hat{\mathcal{C}} \leftarrow \hat{\mathcal{C}} \cup \mathcal{S}_{\text{cur}}$
9: $\quad\quad$ **if** $f^{\text{lex}}(\mathcal{S}_{\text{cur}}) > f^{\text{lex}}(\mathcal{S}_{\text{bsf}})$ **then** $\mathcal{S}_{\text{bsf}} \leftarrow \mathcal{S}_{\text{cur}}$
10: $\quad$ **for** all $\mathcal{D} \in \hat{\mathcal{C}}, \mathcal{D} \notin \mathcal{C}'$
11: $\quad\quad age[\mathcal{D}] \leftarrow 0$
12: $\quad\quad \mathcal{C}' \leftarrow \mathcal{C}' \cup \{\mathcal{D}\}$
13: $\quad \mathcal{S}_{\text{exc}} \leftarrow \mathsf{ApplyExactSolver}(\mathcal{C}', t_{\text{exc}})$
14: $\quad$ **while** $r(\mathcal{S}_{\text{exc}}) = 1$
15: $\quad\quad \mathcal{S}_{\text{exc}} \leftarrow \mathsf{ApplyRepairProcedure}(\mathcal{S}_{\text{exc}})$
16: $\quad$ **if** $f^{\text{lex}}(\mathcal{S}_{\text{exc}}) > f^{\text{lex}}(\mathcal{S}_{\text{bsf}})$ **then** $\mathcal{S}_{\text{bsf}} \leftarrow \mathcal{S}_{\text{exc}}$
17: $\quad (\sigma_{1,t+1}, \ldots, \sigma_{K,t+1}) \leftarrow \mathsf{UpdateRates}((\sigma_{1,t}, \ldots, \sigma_{K,t}), \mathcal{C}', \mathcal{S}_{\text{exc}}, \lambda, \tau)$
18: $\quad \mathcal{C}' \leftarrow \mathsf{Adapt}(\mathcal{C}', \mathcal{S}_{\text{exc}}, age_{\text{limit}})$
19: $\quad t \leftarrow t + 1$
20: **return** $\mathcal{S}_{\text{bsf}} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k\}$

Figure 9.5: Multi-Constructor CMSA for the MDDSP

9.4.5 Algorithmic details

Fig. 9.5 provides the full pseudo-code of our Multi-Constructor CMSA algorithm for the MDDSP. First of all, the weights $(\sigma_{1,t}, \ldots, \sigma_{K,t})$ of the K constructors are initialized to the same value $1/K$, to give them all the same probability of being chosen at the first iteration. Moreover, the sub-instance $\mathcal{C}'$ and the best solution found so far $\mathcal{S}_{\text{bsf}}$ are initialized to empty sets.

The main loop of CMSA starts at line 4, and it is terminated when either a total time limit is reached, or in case a best solution $\mathcal{S}_{\text{bsf}}$ such that $f(\mathcal{S}_{\text{bsf}}) = \delta(G) + 1$ is found. At each CMSA iteration, the CONSTRUCT and MERGE steps are repeated until $n_{\text{sols}}/|V|$ solutions are generated; see lines 5–12. In the construction procedure, the multi-constructor plays a crucial role. First, at line 7 one of the available constructors is chosen by means of a biased random selection. Hereby, constructors are weighted according to their corresponding probabilities. Then, a new solution is generated with

the chosen constructor (line 8). Naturally, every solution $\mathcal{S}_{\text{cur}}$ is compared with the incumbent $\mathcal{S}_{\text{bsf}}$, using the lexicographic objective function presented in Section 9.4.3, as it sometimes may happen that a constructor is able to provide a new best solution. Then, the corresponding MERGE step is performed in lines 10–12. All the new dominating sets found in the $n_{\text{sols}}/|V|$ constructed solutions are added to $\mathcal{C}'$, and their $age[\mathcal{D}]$ is initialized to zero.

Lines 13–17 contain the SOLVE phase. First of all, at line 13, CPLEX solves the set packing based ILP model corresponding to sub-instance $\mathcal{C}'$. The time limit is set to t_{exc} CPU seconds, or to the remaining time budget if this is less than t_{exc} CPU seconds. The output $\mathcal{S}_{\text{exc}}$ is the best solution returned by CPLEX within the given CPU time. In our experiments, we observed that the exact solver is often able to prove the optimality of $\mathcal{S}_{\text{exc}}$ for $\mathcal{C}'$. In other words, often CPLEX does not spend all the allotted computation time. A residual function $r(\mathcal{S}_{\text{exc}})$ is checked at lines 14–15 to verify whether at least one additional dominating can be generated using the vertices not included in any dominating set of $\mathcal{S}_{\text{exc}}$; see Section 9.4.3 for the definition of the function $r()$. If this is the case, a heuristic procedure, labeled ApplyRepairProcedure in Fig. 9.5, is activated iteratively to build the additional dominating set(s), using the vertices in $V \setminus \mathcal{S}_{\text{exc}}$. Then, at line 17, the probabilities of the constructors are reinforced according to the rules presented in Chapter 10.

Finally, the ADAPT phase takes place at line 18. First, the dominating sets $\mathcal{D} \in \mathcal{S}_{\text{exc}}$ generated by the repair procedure that are not already included in the sub-instance $\mathcal{C}'$, are added to $\mathcal{C}'$. Second, the age values of all dominating sets from $\mathcal{S}_{\text{exc}}$ are reset to zero. Third, the age values of all remaining dominating sets from $\mathcal{C}'$ are incremented by one. Finally, all dominating sets $\mathcal{D} \in \mathcal{C}'$ with $age[\mathcal{D}] > age_{\text{limit}}$ are removed from $\mathcal{C}'$.

9.5 Experimental results

We denote the three versions of CMSA studied in this work as CMSA, CMSA-L_{all}, and CMSA-$L_{1,2}$. The first one is the standard CMSA, which only uses one constructor (MDDS-GH_r) and no reinforcement learning mechanism. The second one employs the six constructors described before in this chapter, while the third one uses only the most promising constructors, namely, MDDS-GH_r and IAM_r. We perform a thorough comparison on a large instance space, that involves the two Multi-Constructor CMSA variants against the standard CMSA and the heuristics from the literature. We present and compare also the results obtained by CPLEX.

Our software was implemented in C++17 and compiled on Ubuntu 20.04.5 with g++ 9.4.0 in `-O3` mode. We run all experiments on a machine equipped with an Intel Xeon Processor (Cascadelake), with 16 cores and a clock frequency of 2.4 GHz. We employed a maximum of one single core per experiment, including the call to exact solver within CMSA. For the comparison, we implemented also the six heuristics in C++17 and compiled and ran them under the same experimental setting. Finally, we implemented the ILP model for the MDDSP for CPLEX 20.1, through its C++ interface.

9.5.1 Instances

We consider four kind of graphs:

- Random graphs (RGs)
- Watts-Strogaz networks
- Barabási–Albert networks
- Random geometric graphs (RGGs)

All instance sets include instances with up to 1000 vertices, except for random geometric graph that have up to 5000 vertices. Random graphs (RGs) are general graphs, characterized by two features: the number of vertices, $|V|$, and a density $|D|$. In these graphs, any pair of vertices may be potentially connected. Watts-Strogatz networks, having small-world properties (Watts and Strogatz, 1998), and Barabási–Albert networks, which are scale-free networks (Barabási and Albert, 1999), use custom statistical distributions. Random gGeometric graphs (RGGs), on the other hand, are characterized by $|V|$ and a radius $r_{max} < 1$. $|V|$ vertices are placed at random coordinates (x, y) on the $[0, 1]^2$ square. Then, every pair of vertices $v_i \neq v_j$ with an Euclidean distance $d(v_i, v_j) < r_{max}$ are connected by an edge. These graphs are also known as planar unit disk graphs. RGGs, furthermore, are typically used as a graph model for wireless sensor networks mentioned in Section 9.2. Fig. 9.6 gives a visual representation of the difference between RGs and RGGs. The sets of random graphs, Watts-Strogatz networks and Barabási–Albert networks comprise 60 combinations of $|V|$ and D, with 20 graphs for combination and network type. The set of RGGs comprises 21 combinations of $|V|$ and r_{max} (see Table 9.5). For each of these 21 graph parameter combinations 20 graphs are available.

To generate the combinations of $|V|$ and D to generate random graphs, Watts-Strogatz and Barabási–Albert networks with up to 1000 vertices

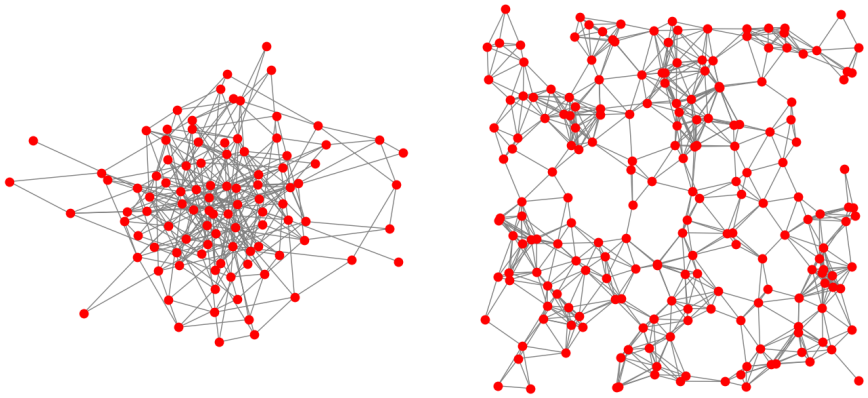


Figure 9.6: Random Graph (left) and Random Geometric Graph (right)

we designed a generation procedure based on the Hammersley point set (Hammersley and Handscomb, 1964), that ensures that all areas of the instance space are equally represented. The instance space is characterized by two dimensions: the number of vertices $|V|$ and the density D. Given an undirected graph $G = (V, E)$, the definition of D is as follows:

$$D = \frac{|E|}{\binom{|V|}{2}} = \frac{2|E|}{|V|(|V|-1)} \tag{9.10}$$

For the three categories of graphs, the problem instances were then generated according to the following procedure:

1. Selection of 60 pseudo-random points $(|V|, D)$ with the Hammersley sampling procedure, with $10 \leq |V| \leq 1000$, and $0.05 \leq D \leq 0.95$. This domain covers practically the whole instance space, avoiding at the same time the generation of graphs that are too small and with density values too extreme. The outcome of the generation procedure is shown graphically in Fig. 9.7. Additionally, information on the 60 points is provided in Tables 9.6 to 9.8, in columns $|V|$ and D.

2. For every point, 20 graphs were randomly generated, resulting in a total of 1200 graphs, for each model. The graphs have $|V|$ vertices, and D is used as the probability for every possible edge to be present. Nonetheless, the distribution of the edges will vary in different kinds of graphs. For random graphs, the graph generator iterates over all pairs of nodes (u, v), and establishes stochastically if an edge between them

exists, with probability D. Watts-Strogatz networks are similar, but in addition, vertices are clustered. In Barabási–Albert networks the degree follows a scale-free (or power law) distribution, that will result in few vertices being much more connected than most other vertices. Given that D is used as input parameter for probabilistic generation, the actual density of the generated graphs can be slightly different from the required one. However, the deviation is minimal: as a double-check, we verified that the actual average density for each group of instances is identical to the required one, if rounded to two decimal digits.

We employed the same procedure also for the generation of an additional set of instances for parameter tuning. This additional set utilizes a higher number of Hammersley points on the same instance space, with of a single instance per point, and is composed exclusively of random graphs. Table 9.2 introduces a classification of the instances depending on their size and density. The borders between regions are visible also in the background grid in Fig. 9.7.

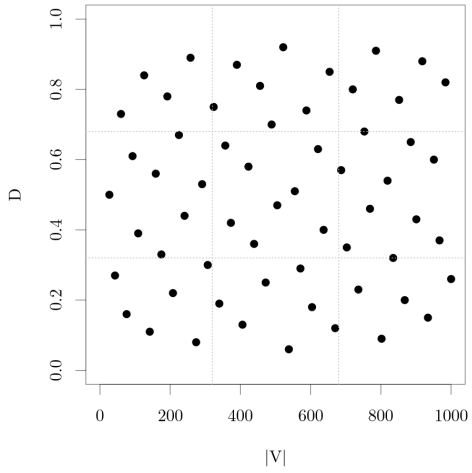


Figure 9.7: The 60 instance groups generated with the Hammersley point set, on the instance space.

The reason for the inclusion of different types of graphs is twofold. On the one hand, random graphs are not realistic in many contexts. In contrast, RGGs, Watts-Strogatz networks and Barabási-Albert networks are often

used in network science, for modelling both natural and social phenomena. On the other hand, we aim to study the adaptability of our CMSA to unseen graphs with different characteristics, where "unseen" refers to the fact that these special types of graphs are not included in the training set for parameter tuning. In this work, RGGs are treated differently from the other graph types, because CMSA rather easily solves problem instances with up to 5000 vertices, even to optimality.

All the instances, used for tuning and for the final experimental evaluation, are available online for download, together with an instance and solution validator, at `https://bitbucket.org/maximum-disjoint-dominating-sets-problem/maximumdisjointdominatingsets-instances`.

Table 9.2: Classification of graphs by size and density

Size		Density	
Name	Feature	Name	Feature
Small	$\|V\| < 334$	Sparse	$D < \frac{1}{3}$
Medium	$334 \leq \|V\| < 667$	Medium	$\frac{1}{3} \leq D < \frac{2}{3}$
Large	$667 \leq \|V\|$	Dense	$\frac{2}{3} \leq D$

9.5.2 Parameter tuning

We performed the parameter tuning with IRACE, which is a tool for automatic algorithm configuration based on iterated racing (López-Ibáñez et al., 2016). The time limit given to the three CMSA variants for each application was $\frac{|V|}{2}$ CPU seconds, identical to the one used for the final experimental evaluation, with the exception of RGG instances, that were allowed $|V|$ CPU seconds. The dependency of running time on instance size allows more time for larger graphs, in which the constructors take longer for the generation of solutions. Tables 9.3 and 9.4 summarize the parameters involved in the tuning, the considered domains, and the outcome of the tuning procedure, respectively for RGGs and other graphs, for the three versions of CMSA. In particular, the tuning procedure considered four CMSA parameters (n_{sols}, d_{rate}, c_{list}, age_{limit}) and the hyperparameters λ and τ, needed for the reinforcement learning procedure. The parameter t_{exc} was also included in the tuning for RGGs, while was fixed to 15 CPU seconds according to the outcome of preliminary exploratory tuning tests for the other graphs. The choice of fixing one parameter in advance aimed at a simplification of the tuning process. Note also that the choice of

setting the number of solution constructions per iteration to $n_{\text{sols}}/|V|$ was determined as being superior to a static setting of the number of solution constructions by the results of a dedicated tuning session with IRACE (this was not done for RGG, though). Parameters d_{rate}, λ and τ are real-valued, with an allowed precision of two digits behind the comma, while n_{sols}, c_{list}, and age_{limit} are natural numbers. We tuned separately the parameters of the three algorithms CMSA, CMSA-L$_{\text{all}}$, CMSA-L$_{1,2}$. The parameter tuning was carried out only on the dedicated training set of RGGs and random graphs, and the tuning budget was set to 5000 experiments for CMSA-L$_{\text{all}}$ and CMSA-L$_{1,2}$, and to 3000 experiments for CMSA, which has fewer parameters. This resulted in a total computation time of 367 hours for CMSA-L$_{\text{all}}$ and CMSA-L$_{1,2}$, and 220 hours for CMSA. Thanks to the availability of a cluster with multiple nodes, having 16 cores each, we could parallelize the experiments, so that the elapsed time needed for the tuning was approximately one day. By contrast, the total computational time needed for the validation was 2590 hours. Less formal preliminary experiments, which are naturally conducted during the design and the implementation of the algorithm, are not accounted in the calculation.

Table 9.3: CMSA parameters, the considered domains for parameter tuning, and the finally determined parameter values for Random geometric graphs.

Parameter	Domain	Value
n_{sols}	$\{2, 3, ..., 30\}$	20
d_{rate}	$[0.60, 1.00]$	0.69
c_{list}	$\{2, 3, ..., 30\}$	22
age_{limit}	$\{2, 3, ..., 30\}$	16
t_{exc}	$\{3, 4, ..., 100\}$	81

9.5.3 Results

Experimental results obtained by CMSA, CMSA-L$_{\text{all}}$, CMSA-L$_{1,2}$ and by the six deterministic greedy heuristics from the literature are displayed in Table 9.5 for Random geometric graphs, Table 9.6 for random graphs, in Table 9.7 for Watts-Strogatz networks and in Table 9.8 for Barabási-Albert networks. Each table row shows averages over 10 independent runs (concerning the CMSA variants) for each of the 20 problem instances produced for the corresponding combination of $|V|$ and r_{max} for RGGs and of $|V|$ and D for the other graph types, in which each row also matches a

Table 9.4: CMSA parameters, the considered domains for parameter tuning, and the finally determined parameter values for random graphs, Watts-Strogatz and Barabási-Albert networks

Parameter	Domain	CMSA	CMSA-L_{all}	CMSA-$L_{1,2}$
n_{sols}	{2500, 100000}	84798	79560	86865
d_{rate}	[0.60, 1.00]	0.99	0.98	0.97
c_{list}	{2, 3, ...,50}	9	3	2
age_{limit}	{2, 3, ...,30}	4	4	11
λ	[0.00, 1.00]	—	0.98	0.30
τ	[0.00, 0.10]	—	0.04	0.08

Hammersley point in the generation procedure. Henceforth, we will call the 20 problem instances belonging to a specific point an instance group. Best values within instance groups are marked in bold. No results in bold in a given row indicates that the best values are found by one of the ILP models.

Results on random geometric graphs

We report the results obtained by CMSA under the parameter setting presented in Table 9.3. CMSA improves on all graphs and finds 4.2 disjoint dominating sets more than the best greedy heuristic (P-Max) for RGGs with $|V| = 5000$ and $r_{max} = 0.3$. The interest in reporting CMSA computational time is to show that it is able to find quite often solutions whose objective function values coincide with the upper bound $\delta(G) + 1$, that is, they are provenly optimal. A group of instances with one or more provenly optimal solutions found by CMSA reports a lower computational time than V CPU seconds, because one or more run terminates before.

Fig. 9.8 presents two graphics relative to the results obtained on RGGs. The first one shows–for each radius–the evolution of the optimality rate depending on the instance size. The second one displays the evolution of the average execution times. Clearly, optimal solutions to these instances coincide much more often with the upper bound. In cases ($|V| = 1500, r_{max} = 0.1$) and ($|V| = 3000, r_{max} = 0.1$), for example, all 20 graphs were solved to proven optimality by CMSA. The algorithm was able to achieve this on average in 77, respectively 3, CPU seconds. The graphics also show that with a growing graph density–a growing value of r_{max}–the values of optimal solutions seem to coincide every time less with the upper bound.

Given that RGGs are solved to optimality rather easily, we don't explore them further and we don't apply the Multi-Constructor variant on them.

Table 9.5: Numerical results obtained for RGGs.

Instances		CMSA		MDDS-GH		IAM		P-MAX		P-MIN		R-LID		COLOR-DDS	
$\|V\|$	r_{max}	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)	obj	t(s)
1000	0.1	**7.90**	80	7.30	0.04	6.30	0.33	7.30	0.02	6.75	0.12	7.15	0.01	7.35	0.01
	0.2	**30.55**	50	29.30	0.12	28.30	0.47	29.95	0.05	27.25	0.12	29.40	0.02	29.35	0.04
	0.3	**71.45**	605	67.00	0.25	66.00	0.84	68.55	0.13	64.20	0.28	68.95	0.03	69.20	0.10
1500	0.1	**12.60**	77	11.65	0.21	10.65	0.94	11.75	0.17	10.60	0.28	11.40	0.18	11.40	0.12
	0.2	**48.00**	298	45.95	0.57	44.95	1.67	45.75	0.27	43.05	0.54	45.85	0.17	46.00	0.20
	0.3	**105.50**	520	101.25	1.62	100.25	2.34	102.55	0.81	95.40	1.08	102.20	0.19	102.70	0.38
2000	0.1	**16.90**	302	15.90	0.27	14.90	1.69	15.95	0.16	14.20	0.27	15.80	0.05	16.00	0.04
	0.2	**62.55**	566	60.25	4.78	59.25	10.09	60.85	1.66	56.55	4.44	60.25	0.08	60.85	0.19
	0.3	**141.75**	1047	135.45	15.21	134.45	23.40	137.35	4.53	128.65	11.19	137.80	0.15	137.90	0.69
2500	0.1	**21.00**	178	20.20	0.79	19.20	3.07	19.85	0.55	18.35	0.77	19.55	0.36	19.85	0.38
	0.2	**79.85**	930	76.70	3.26	75.70	5.63	76.75	1.50	71.40	2.48	76.40	0.42	77.00	0.60
	0.3	**177.30**	1376	169.70	6.51	168.70	9.81	173.45	2.38	160.15	4.57	171.45	0.50	172.80	1.25
3000	0.1	**23.15**	3	22.45	3.38	21.45	9.55	22.35	1.02	20.90	1.92	22.55	0.11	22.55	0.08
	0.2	**93.70**	1383	90.85	31.96	89.85	29.99	89.75	5.09	84.05	15.49	90.70	0.20	91.15	0.51
	0.3	**215.30**	1540	208.05	48.93	207.05	44.92	211.80	14.75	194.85	29.34	208.00	0.27	210.05	2.41
4000	0.1	**33.50**	731	31.35	20.63	30.35	24.59	31.55	4.83	28.90	9.30	31.35	0.22	31.55	0.17
	0.2	**129.45**	1781	125.15	45.65	124.15	48.52	125.70	16.56	117.10	31.75	125.25	0.30	126.30	1.46
	0.3	**283.65**	3282	273.45	63.11	272.45	74.15	279.90	33.42	257.75	45.87	276.65	0.59	278.75	4.80
5000	0.1	**41.30**	730	39.50	31.25	38.50	38.73	39.15	7.40	35.90	19.43	38.80	0.28	39.30	0.31
	0.2	**158.25**	2825	153.65	59.61	152.65	67.34	154.25	23.09	143.50	37.35	153.95	0.56	154.85	2.51
	0.3	**354.55**	4379	342.75	78.34	341.75	98.84	350.35	47.84	321.55	59.36	345.05	0.80	348.20	7.02

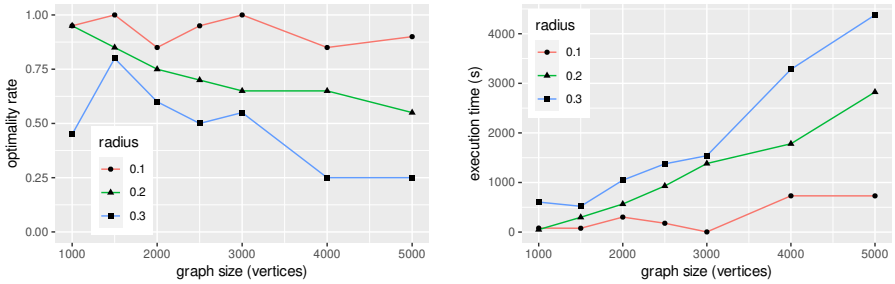


Figure 9.8: Evolution of the optimality rate (left) and execution time (right) depending on the size of the RGG instance, for the three considered radii.

Results on random graphs, Watts-Strogatz and Barabási-Albert networks

Experimental results allow the observation that all three CMSA variants outperform the heuristics on all instance groups. There are, however, some instance groups that appear to be easier to solve, given that all considered CMSA variants obtain exactly the same results. Interestingly, these ties are more frequent in instances with very low or very high density, regardless of their size. A summary of the comparative performances of the algorithms is provided in Table 9.10. CMSA performs very well on all graphs, including those types that were not included in the tuning. Among CMSA variants, the best performer is CMSA-$L_{1,2}$. Interestingly, the advantage of CMSA-$L_{1,2}$ appears to be greater on Watts-Strogatz and Barabási-Albert networks.

We do not report on computational times, because the stopping criterion is the same for all CMSA variants, and the times employed by the six greedy heuristics are negligible, as it is consistently below one second. The total computation times of the CMSA variants range from 13.5 seconds to 500 seconds. Sometimes, however, a CMSA run is stopped before the computation time limit is reached, due to finding a provenly optimal solution. It can be said that the execution times of CMSA are about two orders of magnitude higher than the ones of the greedy heuristics. While this–at first sight–might result in an unfair comparison, we need to consider that greedy algorithms are simple procedures and that their speed advantage comes at the cost of much worse solutions. On the other hand, more sophisticated algorithms such as metaheuristics perform a much larger exploration of the solution space to find better solutions. Inevitably, they require a higher computational time. In our context, longer running times are fully justified by the obtained improvements over the greedy heuristics, which is indisputable on all instance groups and all graph categories. Indeed, the table column with the heading "Improvement (%)" in Tables 9.6 to 9.8, shows the percentage gap between the best-performing CMSA and the best heuristic. The improvement is consistent for all instance groups. In particular, CMSA can find up to 30% more dominating sets in the smallest graphs. The relative improvement decreases as the graph size increases. The difference in terms of the absolute values, however, as shown in column "Improvement (n)", is remarkable, especially on dense graphs, with CMSA able to produce up to 14.42 more dominating sets on random graphs, and up to 13.55 more dominating sets on Watt-Strogatz networks. This is quite a notable result if we consider, for instance, the application to WSNs. Manufacturers of commercial sensor nodes declare operational lifetimes that range from days to

months, or even years (Mak and Seah, 2009). Depending on the application area of a WSN, the potential lifetime of the network is obtained in relation to the number of disjoint dominating sets found on the graph. So, an extension of even a few percentage points in the lifetime of the WSN is translated into a gain of, at least, some days of autonomous functioning.

Given that the running time granted to CMSA is much larger than the time needed by the greedy algorithms, the reader might wonder whether a heuristic algorithm for solving the MDDSP which is granted a comparable execution time would be able to compete with CMSA. However, all existing heuristics for the MDDSP are deterministic greedy algorithms, and non-deterministic metaheuristics have never been applied to the MDDSP. What can be done, however, is to run the best one of our randomized greedy heuristics (MDDS-GH$_r$) in a repeated way with the same computation time limit as used for CMSA, that is, with a time limit of $|V|/2$ seconds. This was done with the same values for parameters d_{rate} and c_{list} found in Table 9.4 in column CMSA. The output of this repeated, randomized greedy heuristic is the best solution found within $|V|/2$ seconds. Finally, we compare the obtained results with the ones of our best-performing CMSA variant (CMSA-L$_{1,2}$).

An analysis of the results shows that, in random graphs, CMSA-L$_{1,2}$ obtains on average 0.97 more dominating sets, which corresponds to an average percentage increment of 0.74%. In the context of Watt-Strogatz networks, CMSA-L$_{1,2}$ obtains on average 0.92 more dominating sets, corresponding to an average percentage increment of 0.72%. However, in the case of Barabási-Albert networks, the gap is significantly reduced, as CMSA-L$_{1,2}$ obtains on average just 0.05 more dominating sets, or, equivalently, an average percentage increment of 0.15%. This is not a surprising fact, considering the analogies between Barabási-Albert networks and random geometric graphs, that were studied in our previous work (Rosati et al., 2023a). Both are, indeed, characterized by locality and/or clustering of nodes. Moreover, we performed three paired Wilcoxon signed rank tests with continuity correction. The tests return that the difference in performance between CMSA-L$_{1,2}$ and the randomized, repeated greedy heuristic is statistically significant for random graphs and Watts-Strogatz networks, given the very low p-values ($< 2.2 \cdot 10^{-16}$). On the other hand, in the case of Barabási-Albert network, even though CMSA-L$_{1,2}$ generally produces better results, the gap is not statistically significant (the p-value is 0.079, above the significance threshold of 0.05), which means that we cannot reject the null hypothesis of equality of the true means of the two populations. These results suggest that the advantage of CMSA in the case

of Barabási-Albert networks is limited to dense graphs (see Fig. 9.13), while its superiority is indisputable in the context of the other graph types.

We point out that MDDS-GH$_r$, that we used for comparison, exists only because we developed it for its usage within CMSA, and that it can be seen as a particular use case of CMSA, with extreme parameter values such as a computation time of zero for solving the ILP model at each iteration. In the context of real-world applications, greedy algorithms are interesting because of their speed and usefulness in time-critical applications in which a user needs a response from the computer in fractions of seconds. In less time-critical applications, in which the decision maker has more time available for finding a good solution in advance, repeating in a loop the same randomized greedy algorithm for some time is generally not the best practice, especially if a metaheuristic (such as CMSA) that offers better exploration of the search space in the same running time exists.

In order to get more information on the behavior of CMSA in different regions of the instance space, Fig. 9.9 shows six plots. All plots show the instance space, with the two axes that correspond to $|V|$ and D. The three plots from the top row indicate those instance groups (out of 60 Hammersley points) for which, from left to right, CMSA, CMSA-L$_{\text{all}}$, and CMSA-L$_{1,2}$ obtain the best results. The plots consider only random graphs, although similar considerations could be drawn also for the other kinds of instances. In the three plots of the bottom row, the easy instances where all methods are able to obtain the same average results have been eliminated. Such instances are mostly concentrated in the low end or in the high end of the density range. From the plots, we can observe that CMSA performs well for small and medium instances. On the other side, it seems to lose its effectiveness in the context of larger instances, for which CMSA-L$_{1,2}$ obtains the best solutions. Beyond the graphical impression, this is confirmed with statistical significance by a non-parametric test, which is discussed in Section 9.5.4. On the other hand, Algorithm CMSA-L$_{\text{all}}$ appears to be particularly weak on very dense graphs. To provide the reader with the complete picture, Fig. 9.10 represent the best performers in a single plot, by using the same three symbols for the three algorithms as those already used in Fig. 9.9.

ILP model results

Table 9.9 displays the results obtained from the solution of the three models ILP, ILP-SM$_1$, and ILP-SM$_2$ presented in Section 9.3, applied to all problem instances with $|V| < 300$, within a computation time limit of one hour. The reason why we only consider instances with $|V| < 300$ is twofold: on the

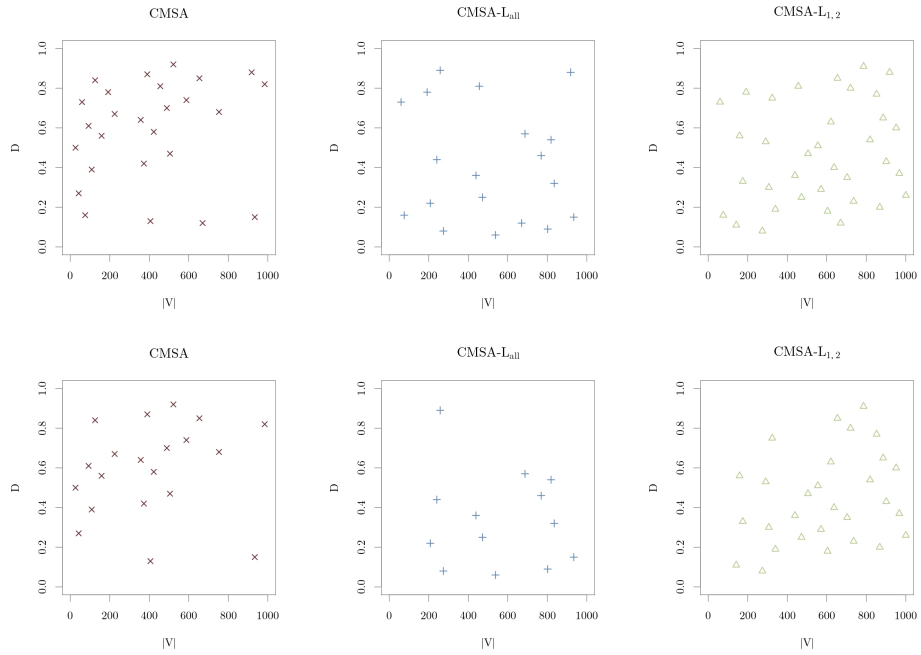


Figure 9.9: Instance groups (Hammersley points) for which algorithms are the best-performing ones. The plots in the top row include all instances, while in the plots of the bottom row, the six points in which all CMSA variants perform equally well are removed.

one hand, the exact solver struggles in finding even any feasible solution for graphs of sizes slightly above 200 vertices. On the other hand, the application of CPLEX is quite memory intensive in the context of larger instances. Running such experiments would be, in any case, impractical. The table is organized as follows: for the three network types (RG: Random graphs, WS: Watts-Strogatz Network and BA: Barabási-Albert networks), and for the three ILP models (ILP: no symmetry breaking constraints, ILP-SM$_1$ and ILP-SM$_2$: the two models that implement symmetry breaking constraints), for every instance group the average objective function value, the execution time and the ratio of proven optimal solutions are provided. Bold values are used to mark those results which are of at least the same quality as the ones of CMSA. Indeed, there are some instances for which the ILP results are better than the ones of CMSA and the heuristics. This happens mainly on the smallest graphs ($|V| < 100$) and on very sparse graphs ($D < 0.1$). Usually, ILP solving requires much longer running time than CMSA, which

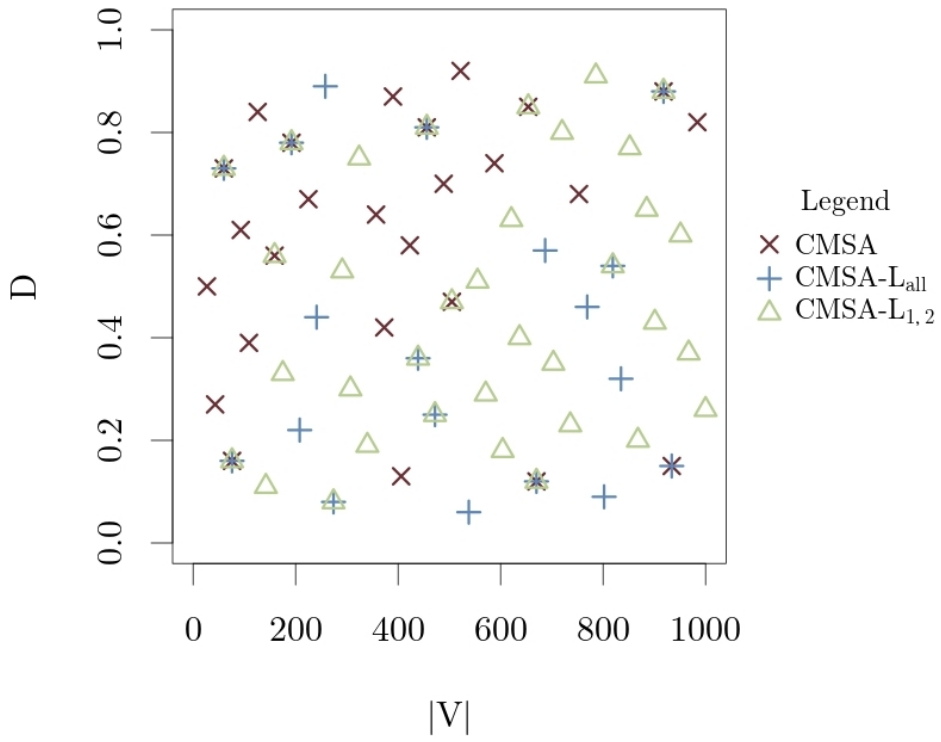


Figure 9.10: Summary of the results shown in Fig. 9.9 in a single plot.

is limited to $|V|/2$ CPU seconds, or the greedy heuristics which take less than a few seconds. There are a few cases, however, in which the ILP solver quickly finds a provenly optimal solution. This happens in the case of random graphs and for Barabási-Albert networks, but never for Watts-Strogatz ones. Interestingly, we do not observe any improvements due to the application of symmetry breaking constraints. The reasons for this might be twofold. First, modern commercial solvers like CPLEX already contain the implementation of techniques for detecting and handling symmetries, even when this is not explicitly specified in the model. Second, the reduction of the search space caused by symmetry breaking is not enough to outweigh the increased complexity of the model within the given computation time limit of one hour. The application of CPLEX to our ILP model results in provenly optimal solutions for a total of 79 random graphs, 47 Watts-Strogatz networks, and 60 Barabási-Albert networks. We compare these solutions with the respective 790, 470 and 600 solutions found by CMSA-L$_{1,2}$ in the 10 runs executed

on each graph. On random graphs, CMSA-$L_{1,2}$ is able to find the optimal solution, with identical cost as the one found by the ILP Model, in 648 out of 790 runs, with an average gap of 2.37%, computed on the total runs. On Watts-Strogatz networks, CMSA-$L_{1,2}$ finds the optimal solution in 284 out of 470 runs with an average gap of 4.88%. Finally, on Barabási-Albert networks, CMSA-$L_{1,2}$ finds the optimal solution in 302 out of 600 runs, with an average gap of 8.34%.

9.5.4 Statistical analysis

Additionally, `R` package `scmamp` (Calvo and Santafé Rodrigo, 2016) was used to facilitate and support the interpretation of the results from a statistical point of view. For this purpose, first, the results of the considered algorithms are compared simultaneously using the Friedman test for obtaining the rejection of the hypothesis that they all perform equally. Next, corresponding pairwise comparisons are performed using the Nemenyi post-hoc test (Garcia and Herrera, 2008) and, eventually, the output of this statistical analysis is presented by means of *critical difference* (CD) plots. The CD plot that compares all heuristic algorithms (CMSA variants and greedy heuristics) on all 3600 random, Watts-Strogatz and Barabási-Albert graphs together is shown in Fig. 9.11. The horizontal axis of a CD plot represents the range of algorithm ranks, while each of the vertical lines represents the average rank of the corresponding algorithm. Bold horizontal lines connecting algorithm markers indicate that the corresponding algorithms perform statistically equivalent–i.e. the critical difference is not greater than the significance level of 0.05–concerning the considered set of problem instances.

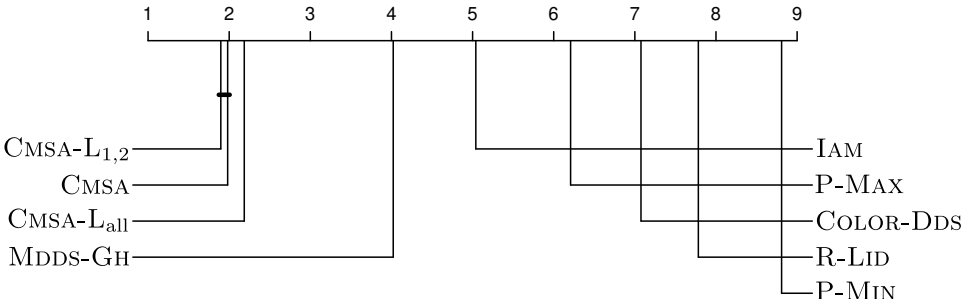


Figure 9.11: Critical difference plot comparing all heuristic algorithms on all 3600 problem instances.

The CD plot from Fig. 9.11 shows the following. First, the three CMSA variants outperform, with statistical significance, all six greedy heuristics,

on the whole set of graphs. Second, the performance differences between all pairs of greedy heuristics are statistically significant. Third, CMSA-$L_{1,2}$ performs better–with statistical significance–than CMSA-L_{ALL}. Fourth, even though the average ranking of CMSA-$L_{1,2}$ is better than the one of CMSA and the average ranking of CMSA is better than the one of CMSA-L_{ALL}, no statistically significant difference can be found between CMSA-$L_{1,2}$ and CMSA.

Finally, we repeated the CD plot analysis limiting the set of considered instances to small, medium and large-sized graphs. The resulting CD plots (only concerning the CMSA variants) can be found in Fig. 9.12. They show that, while no statistical difference can be found between CMSA-$L_{1,2}$ and CMSA in the context of small graphs, CMSA-$L_{1,2}$ is found to outperform CMSA with statistical significance in the context of and medium-sized and large graphs. This confirms our observations from the numerical results (Tables 9.6 to 9.8), and in part, from the analysis provided in Figs. 9.9 and 9.10.

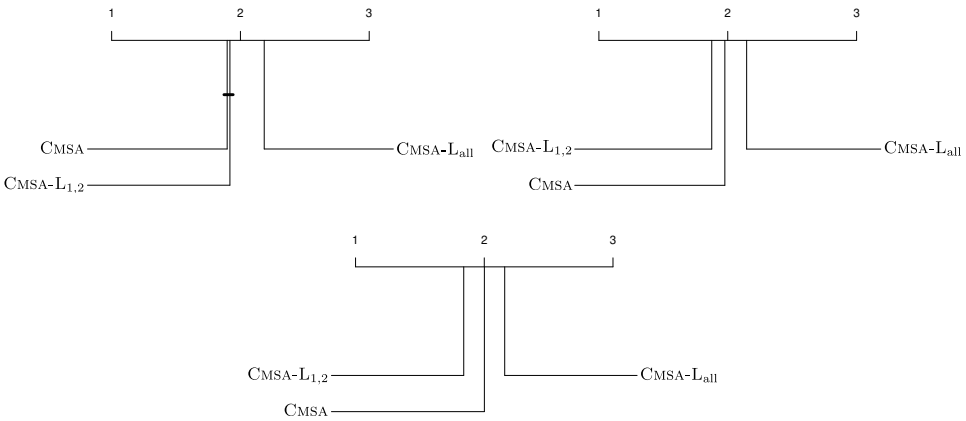


Figure 9.12: Comparison depending on graph size. Top left: small graphs, top right: medium-sized graphs, bottom: large graphs.

In addition to the statistical comparison, in Fig. 9.13 we show the absolute improvements obtained by CMSA-$L_{1,2}$, which is the best performing CMSA, over the best greedy result. This is done for the three considered types of graphs. As we commented previously, the improvement is more noticeable on graphs with higher densities. For this reason, the graph density is shown on the x-axis, while the y-axis shows the improvement, measured as the difference between the dominating sets found by CMSA-$L_{1,2}$ and the dominating sets in the best greedy result. For each density point, a boxplot

shows the distribution of the obtained improvements.

9.6 Conclusions

In this chapter, we proposed the Multi-Constructor CMSA for solving the Maximum Disjoint Dominating Sets Problem. This algorithm uses a stochastically biased selection criterion to choose the constructors and a reinforcement learning technique for learning the best probabilities for the constructors during the search process.

The standard CMSA and two variants of the Multi-Constructor CMSA were tested: the first one, CMSA-L_{all}, makes use of all available constructors, while the second one, CMSA-$L_{1,2}$, only utilizes the two most promising constructors.

We were able to show that all our CMSA variants outperform the greedy heuristics from the related literature on four distinct models of graphs: random, random geometric, Watts-Strogatz and Barabási-Albert. Moreover, CMSA-$L_{1,2}$ generally performs best, with a statistically significant advantage over the other two CMSA variants in the context of medium-sized and large graphs, which are the most challenging ones.

We also made an attempt to solve the MDDSP directly by means of applying a MIP solver to three ILP models, two of which implement symmetry breaking constraints. However, this did not yield satisfactory results, with the exception of very sparse graphs.

Overall, CMSA-$L_{1,2}$ performs best on 116 out of 180 instance groups, which corresponds to 64.4% of the instances. When considering all CMSA variants together, they perform best on 94.4% of the instances, that is, 170 out of 180 instance groups. The remaining 10 instance groups are small and sparse graphs for which the ILP models obtains better results in a reasonable time limit.

We executed all the experiments on a set of graphs that we have generated from an instance space spanned by graph size (number of vertices) and graph density. This was done with a statistically sound procedure that guarantees that all regions of the instance space are represented in the sample.

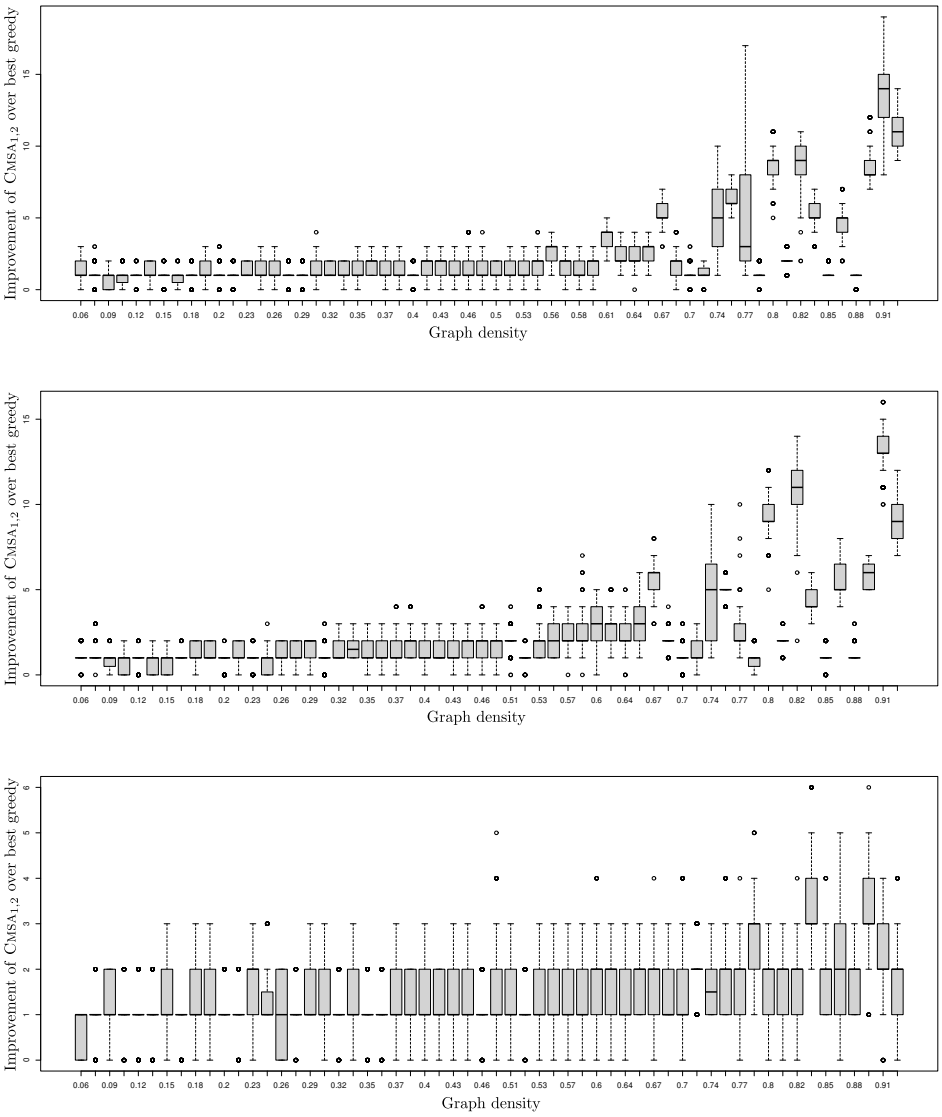


Figure 9.13: Improvement obtained by CMSA-$L_{1,2}$ over the best greedy result, for instance groups, ordered by increasing density. From top to bottom: random graphs, Watts-Strogatz networks, Barabási-Albert networks.

Table 9.6: Numerical results obtained for random graphs

Instances		CMSA			Heuristics						Improvement	
$\|V\|$	D	CMSA	CMSA-L_{ALL}	CMSA-$L_{1,2}$	MDDS-GH	IAM	P-MAX	P-MIN	R-LID	COLOR-DDS	(%)	(n)
27	0.50	**8.00**	7.96	7.96	6.20	6.10	5.75	4.90	5.30	5.55	29.03	1.80
43	0.27	6.32	6.26	6.26	5.20	4.40	4.65	3.95	4.25	4.40	21.54	1.12
60	0.73	**23.00**	**23.00**	**23.00**	21.50	20.95	18.10	15.30	16.40	16.50	6.98	1.50
76	0.16	5.70	5.70	5.70	4.80	4.00	4.15	3.75	3.95	3.95	18.75	0.90
93	0.61	**28.89**	28.72	28.80	25.20	24.20	19.90	16.65	18.05	18.45	14.64	3.69
109	0.39	**19.25**	19.13	19.22	17.40	16.95	13.70	11.30	12.60	12.90	10.63	1.85
126	0.84	**60.12**	60.02	60.04	54.35	53.75	43.20	36.40	38.75	39.80	10.62	5.77
142	0.11	6.78	6.80	6.82	6.10	5.15	5.05	4.15	4.55	4.60	11.80	0.72
159	0.56	**39.03**	38.83	**39.03**	36.05	35.60	28.00	23.10	25.50	25.75	8.27	2.98
175	0.33	23.76	23.77	**23.78**	22.30	21.55	17.00	13.95	15.65	15.75	6.64	1.48
192	0.78	**64.10**	**64.10**	**64.10**	62.65	61.95	52.30	43.65	48.15	48.65	2.31	1.45
208	0.22	18.79	**18.82**	18.81	17.65	16.95	13.60	10.90	12.10	12.35	6.63	1.17
225	0.67	**65.98**	65.54	65.77	59.85	59.30	46.90	38.65	42.60	43.65	10.24	6.13
241	0.44	41.16	**41.20**	41.18	39.35	38.90	29.60	24.65	27.25	27.50	4.70	1.85
258	0.89	128.28	**128.30**	128.26	118.65	118.85	92.00	75.45	81.95	82.80	7.95	9.45
274	0.08	9.06	**9.11**	**9.11**	8.25	7.35	6.65	5.20	6.00	6.05	10.42	0.86
291	0.53	57.78	57.76	**57.82**	55.95	55.55	42.80	35.15	39.20	39.35	3.34	1.87
307	0.30	34.53	34.54	**34.61**	33.25	32.35	24.60	20.05	22.35	22.85	4.09	1.36
324	0.75	107.99	107.86	**108.00**	101.30	100.90	76.70	62.90	69.60	70.05	6.61	6.70
340	0.19	24.94	24.94	**24.98**	23.75	22.75	17.60	14.10	15.70	16.15	5.18	1.23
357	0.64	**87.93**	87.84	87.82	85.30	84.65	64.90	53.05	58.70	59.50	3.08	2.63
373	0.42	**56.77**	56.76	56.75	54.90	54.20	40.85	33.20	37.00	37.40	3.41	1.87
390	0.87	148.51	147.96	**148.27**	142.90	142.65	120.05	100.00	110.75	111.85	3.93	5.61
406	0.13	**20.67**	20.63	20.66	19.45	18.60	14.40	11.45	12.95	13.15	6.27	1.22
423	0.58	**87.44**	87.42	87.42	85.30	85.10	65.15	53.45	59.40	60.00	2.51	2.14
439	0.36	55.41	**55.44**	**55.44**	53.55	52.95	39.65	31.85	35.80	36.35	3.53	1.89
456	0.81	**152.00**	**152.00**	**152.00**	149.35	149.05	118.25	97.05	107.20	107.85	1.77	2.65
472	0.25	41.93	**41.94**	**41.94**	40.35	39.65	29.55	23.75	26.75	26.90	3.94	1.59
489	0.70	**122.64**	122.59	122.59	121.15	120.35	96.10	79.35	88.35	88.65	1.23	1.49
505	0.47	**81.16**	81.11	**81.16**	79.40	78.80	58.80	47.90	53.50	54.25	2.22	1.76
522	0.92	**260.79**	260.74	260.72	246.45	248.60	188.15	152.70	167.25	168.80	4.90	12.19
538	0.06	12.93	**12.97**	12.96	11.80	10.75	9.05	7.30	8.05	8.30	9.92	1.17
555	0.51	95.84	95.84	**95.90**	93.65	93.50	70.40	57.10	63.65	64.50	2.40	2.25
571	0.29	56.06	56.10	**56.11**	54.55	54.10	39.80	31.90	36.00	36.40	2.86	1.56
588	0.74	**160.84**	157.96	160.07	154.55	154.25	122.45	101.35	112.80	113.50	4.07	6.29
604	0.18	38.03	38.03	**38.04**	36.95	36.00	26.75	21.40	23.95	24.40	2.95	1.09
621	0.63	137.00	137.06	**137.16**	133.95	133.85	100.95	82.75	92.10	93.05	2.40	3.21
637	0.40	84.15	84.16	**84.18**	82.40	82.05	60.05	48.50	54.95	55.10	2.16	1.78
654	0.85	**218.00**	217.99	**218.00**	216.50	216.00	178.40	147.00	163.30	164.60	0.69	1.50
670	0.12	**29.00**	**29.00**	**29.00**	27.90	27.15	20.25	15.95	18.15	18.65	3.94	1.10
687	0.57	132.83	**132.84**	132.82	130.35	130.30	96.50	78.50	87.80	89.05	1.91	2.49
703	0.35	79.97	79.94	**79.99**	78.30	77.60	57.25	45.80	51.60	51.95	2.16	1.69
720	0.80	239.26	236.78	**239.36**	228.25	229.70	171.20	139.10	154.95	156.35	4.21	9.66
736	0.23	55.96	55.98	**56.00**	54.65	53.65	39.50	31.30	35.60	35.80	2.47	1.35
753	0.68	**182.77**	182.72	182.74	179.35	179.90	132.80	109.30	121.55	122.40	1.60	2.87
769	0.46	113.27	**113.32**	113.28	110.95	110.80	81.95	66.20	74.75	75.20	2.14	2.37
786	0.91	324.62	322.62	**324.92**	308.60	310.50	250.95	208.75	233.25	234.40	4.64	14.42
802	0.09	26.02	**26.03**	26.02	25.30	24.45	18.20	14.70	16.55	16.85	2.89	0.73
819	0.54	141.38	**141.43**	**141.43**	138.80	138.65	103.85	84.85	95.05	96.05	1.89	2.63
835	0.32	84.58	**84.63**	84.62	83.00	82.20	59.95	48.15	54.50	54.85	1.96	1.63
852	0.77	233.58	232.53	**234.84**	228.65	229.00	181.30	149.05	166.55	167.60	2.55	5.84
868	0.20	56.54	56.56	**56.60**	55.30	54.55	39.85	31.80	35.55	36.20	2.35	1.30
885	0.65	189.34	189.38	**189.47**	186.55	185.80	142.35	116.80	130.80	131.45	1.57	2.92
901	0.43	121.50	121.50	**121.55**	119.45	119.00	87.10	70.65	79.20	79.80	1.76	2.40
918	0.88	**306.00**	**306.00**	**306.00**	305.05	304.20	259.40	215.60	238.60	241.05	0.31	0.95
934	0.15	**46.72**	**46.72**	46.70	45.35	44.80	32.50	25.90	29.30	29.85	3.02	1.37
951	0.60	185.90	185.92	**185.94**	183.35	183.30	135.95	110.65	123.70	125.20	1.41	2.59
967	0.37	110.25	110.22	**110.32**	108.60	108.00	78.50	63.45	71.80	72.00	1.58	1.72
984	0.82	**327.00**	323.32	326.99	315.35	317.60	235.55	192.65	213.80	216.10	2.96	9.40
1000	0.26	80.76	80.78	**80.80**	79.35	78.65	56.90	45.70	51.45	52.00	1.83	1.45

Table 9.7: Results obtained for Watts-Strogatz graphs.

Instances		CMSA			Heuristics						Improvement	
$\|V\|$	D	CMSA	CMSA-L$_{\text{ALL}}$	CMSA-L$_{1,2}$	MDDS-GH	IAM	P-MAX	P-MIN	R-LID	COLOR-DDS	(%)	(n)
27	0.52	**9.00**	**9.00**	8.96	7.95	7.20	6.70	5.80	6.15	6.15	13.21	1.05
43	0.28	7.51	7.40	7.46	6.05	5.15	5.25	4.80	4.60	5.10	24.13	1.46
60	0.73	**24.20**	**24.20**	**24.20**	22.85	22.05	18.45	15.85	16.60	17.55	5.91	1.35
76	0.16	7.00	7.00	7.00	5.90	5.25	5.15	4.05	4.50	4.80	18.64	1.10
93	0.60	**29.67**	29.34	29.36	25.60	24.70	20.10	16.60	17.55	19.10	15.90	4.07
109	0.39	**20.60**	20.38	20.59	18.60	17.65	14.10	11.55	12.05	13.80	10.75	2.00
126	0.84	**61.26**	61.19	61.12	56.55	55.65	43.80	36.55	39.00	40.15	8.33	4.71
142	0.11	8.26	8.17	8.26	7.60	6.70	5.80	5.00	5.05	5.90	8.68	0.66
159	0.57	**39.98**	39.75	39.85	37.45	36.45	28.45	23.85	24.50	27.10	6.76	2.53
175	0.33	25.80	25.75	**25.84**	24.30	23.30	17.20	14.85	14.80	17.65	6.34	1.54
192	0.78	**64.45**	**64.45**	**64.45**	63.50	62.55	51.80	44.45	47.50	49.45	1.50	0.95
208	0.22	20.75	20.78	**20.84**	19.45	18.60	13.35	11.70	11.75	13.90	7.15	1.39
225	0.67	**66.88**	66.48	66.67	61.10	59.95	46.85	38.65	40.85	43.95	9.46	5.78
241	0.44	43.58	43.62	**43.65**	42.00	41.10	29.55	24.70	25.35	29.30	3.93	1.65
258	0.89	**127.61**	127.60	127.60	121.70	120.85	89.40	77.65	81.15	84.05	4.86	5.91
274	0.08	**11.00**	10.99	10.99	9.85	8.85	7.20	6.15	6.35	7.45	11.68	1.15
291	0.53	59.94	59.76	**60.08**	58.10	57.15	43.10	35.50	36.70	41.55	3.41	1.98
307	0.30	37.09	37.08	**37.11**	36.10	34.95	24.60	20.65	20.85	25.25	2.80	1.01
324	0.75	**108.00**	**108.00**	**108.00**	103.00	102.10	76.50	63.75	68.00	70.75	4.85	5.00
340	0.19	**27.00**	**27.00**	**27.00**	25.65	24.70	17.15	14.90	14.80	18.05	5.26	1.35
357	0.64	88.17	88.14	**88.24**	86.15	85.25	64.65	53.15	55.50	60.60	2.43	2.09
373	0.42	**59.08**	**59.08**	**59.08**	57.65	56.65	40.80	33.50	33.80	39.95	2.48	1.43
390	0.87	**157.70**	157.66	157.66	152.10	151.25	119.20	102.80	109.55	112.45	3.68	5.60
406	0.13	**22.38**	22.30	22.35	21.50	20.50	14.10	12.70	12.30	15.00	4.09	0.88
423	0.58	91.30	91.14	**91.36**	89.15	88.10	66.25	54.00	55.75	62.15	2.48	2.21
439	0.36	**59.02**	**59.02**	**59.02**	57.75	56.75	39.85	32.75	33.00	39.70	2.20	1.27
456	0.81	**152.00**	**152.00**	**152.00**	150.05	149.05	117.40	98.65	106.00	108.30	1.30	1.95
472	0.25	45.15	45.15	**45.19**	44.25	43.20	29.35	24.80	24.65	30.25	2.12	0.94
489	0.70	**123.08**	122.84	122.89	121.85	120.85	95.20	78.80	83.25	88.80	1.01	1.23
505	0.47	84.75	84.80	**84.84**	83.15	82.05	58.85	48.60	48.95	57.60	2.03	1.69
522	0.92	**258.32**	258.14	258.08	249.00	248.10	173.70	153.80	159.35	165.75	3.74	9.32
538	0.06	**14.90**	14.87	14.89	13.75	12.80	9.30	8.05	8.10	9.95	8.36	1.15
555	0.51	**102.17**	102.14	**102.17**	100.15	99.10	70.70	57.40	58.45	68.25	2.02	2.02
571	0.29	**60.88**	60.85	60.87	59.35	58.40	39.35	32.70	33.10	40.65	2.58	1.53
588	0.74	162.66	160.50	**162.88**	158.05	157.15	122.35	101.65	108.65	113.55	3.06	4.83
604	0.18	41.73	41.60	**41.78**	40.45	39.35	26.35	22.50	22.20	27.70	3.29	1.33
621	0.63	141.70	141.74	**141.82**	139.25	138.25	101.25	82.55	86.65	94.90	1.85	2.57
637	0.40	88.08	88.10	**88.13**	86.75	85.70	59.95	49.40	49.55	59.25	1.59	1.38
654	0.85	**218.05**	218.02	**218.05**	216.95	216.20	175.80	149.05	159.95	164.40	0.51	1.10
670	0.12	**32.00**	**32.00**	**32.00**	31.10	29.95	19.50	17.10	16.85	21.15	2.89	0.90
687	0.57	134.88	134.85	**134.92**	133.20	132.15	96.55	78.55	80.65	91.80	1.29	1.72
703	0.35	**85.08**	85.06	**85.08**	83.75	82.70	56.65	46.70	46.65	56.90	1.59	1.33
720	0.80	239.35	235.42	**239.61**	230.40	229.60	169.25	141.45	151.80	155.55	4.00	9.21
736	0.23	**61.26**	61.20	61.17	60.25	59.25	38.45	33.05	32.75	40.45	1.68	1.01
753	0.68	183.98	183.97	**184.00**	182.00	181.00	132.50	108.35	114.20	122.50	1.10	2.00
769	0.46	120.84	120.78	**120.88**	119.00	118.00	82.10	66.70	67.45	79.70	1.58	1.88
786	0.91	338.67	**338.70**	338.38	325.15	324.20	240.10	212.50	220.75	228.90	4.17	13.55
802	0.09	**29.00**	**29.00**	**29.00**	28.20	27.15	17.50	15.25	15.35	19.40	2.84	0.80
819	0.54	149.58	**149.60**	**149.60**	147.45	146.45	104.55	84.85	86.55	100.00	1.46	2.15
835	0.32	91.03	91.03	**91.04**	89.55	88.60	59.50	49.05	49.10	60.70	1.66	1.49
852	0.77	235.08	235.07	**235.25**	232.70	231.75	180.80	149.65	160.90	166.35	1.10	2.55
868	0.20	62.03	**62.04**	62.02	61.25	60.20	38.20	33.05	33.05	41.05	1.29	0.79
885	0.65	196.08	196.07	**196.18**	193.30	192.30	142.70	116.30	121.05	132.75	1.49	2.88
901	0.43	**126.81**	126.78	126.78	125.25	124.25	86.35	70.65	71.30	85.80	1.25	1.56
918	0.88	307.71	307.64	**307.74**	306.50	305.65	251.65	218.95	232.20	238.50	0.40	1.24
934	0.15	51.18	51.16	**51.24**	50.40	49.30	30.95	27.30	27.00	33.80	1.67	0.84
951	0.60	187.07	187.06	**187.10**	185.60	184.55	135.70	110.15	114.00	127.75	0.81	1.50
967	0.37	**117.39**	117.31	**117.39**	115.90	114.95	77.70	64.05	64.20	78.45	1.29	1.49
984	0.82	326.90	319.93	**327.14**	316.30	315.50	232.70	194.45	209.80	213.65	3.43	10.84
1000	0.26	**88.00**	**88.00**	**88.00**	86.80	85.75	55.05	46.95	46.45	58.30	1.38	1.20

Table 9.8: Results obtained for Barabási-Albert graphs.

Instances		CMSA			Heuristics						Improvement	
$\|V\|$	D	CMSA	CMSA-L$_{\text{ALL}}$	CMSA-L$_{1,2}$	MDDS-GH	IAM	P-MAX	P-MIN	R-LID	COLOR-DDS	(%)	(n)
27	0.52	**7.00**	**7.00**	6.91	5.80	5.00	5.00	4.40	4.80	4.75	20.69	1.20
43	0.28	5.54	5.48	5.60	4.55	3.75	3.85	3.40	3.80	3.85	23.08	1.05
60	0.73	**17.58**	17.41	17.32	15.25	14.20	11.85	10.55	11.10	11.70	15.28	2.33
76	0.16	4.97	4.96	4.97	4.15	3.65	3.40	3.05	3.55	3.45	19.76	0.82
93	0.60	**19.77**	19.52	19.68	17.65	16.85	13.35	11.85	12.55	13.00	12.01	2.12
109	0.39	14.04	13.98	14.01	12.80	11.85	9.85	8.60	8.90	9.60	9.69	1.24
126	0.84	**36.21**	35.88	36.08	32.65	31.70	23.50	21.05	22.30	23.05	10.90	3.56
142	0.11	5.85	5.87	5.89	4.85	4.35	4.25	3.80	3.90	3.95	21.44	1.04
159	0.57	27.12	27.08	**27.15**	25.80	24.85	18.65	16.40	17.40	17.90	5.23	1.35
175	0.33	17.67	17.72	**17.74**	16.45	15.65	12.05	10.65	11.40	11.55	7.84	1.29
192	0.78	**45.19**	44.80	45.12	42.30	41.40	29.90	26.35	28.40	28.50	6.83	2.89
208	0.22	**14.00**	13.99	**14.00**	13.00	12.05	9.55	8.20	9.05	9.40	7.69	1.00
225	0.67	42.35	42.39	**42.46**	40.85	39.85	28.50	25.40	27.10	27.50	3.94	1.61
241	0.44	29.47	29.38	**29.50**	28.10	27.20	19.75	17.40	18.90	18.80	4.98	1.40
258	0.89	**66.42**	65.96	66.31	62.90	61.95	43.30	39.35	41.70	42.25	5.60	3.52
274	0.08	7.00	7.00	7.00	6.15	5.60	4.85	4.20	4.55	4.90	13.82	0.85
291	0.53	41.05	**41.06**	**41.06**	39.70	38.85	27.45	24.30	25.90	26.25	3.43	1.36
307	0.30	**25.00**	**25.00**	**25.00**	23.70	22.70	16.75	14.50	15.65	16.05	5.49	1.30
324	0.75	64.67	64.67	**64.78**	63.05	62.05	42.85	38.65	40.75	41.45	2.74	1.73
340	0.19	17.88	17.88	**17.89**	16.60	15.75	11.95	10.25	11.20	11.75	7.77	1.29
357	0.64	58.57	58.56	58.70	57.30	56.35	39.00	34.50	37.15	37.25	2.44	1.40
373	0.42	**39.98**	39.97	39.95	38.65	37.80	26.75	23.25	25.20	25.45	3.44	1.33
390	0.87	88.18	88.18	**88.25**	86.20	85.20	58.55	53.40	56.50	56.85	2.38	2.05
406	0.13	**14.40**	14.39	**14.40**	13.40	12.75	9.85	8.25	9.15	9.60	7.46	1.00
423	0.58	61.06	61.05	**61.09**	59.85	58.75	40.40	36.10	38.10	38.75	2.07	1.24
439	0.36	39.20	39.15	**39.24**	38.20	37.20	26.15	22.80	24.60	25.10	2.72	1.04
456	0.81	92.93	92.89	**92.94**	91.10	90.10	61.40	55.15	58.20	58.90	2.02	1.84
472	0.25	**30.00**	29.99	**30.00**	28.65	27.85	20.00	17.00	18.65	18.95	4.71	1.35
489	0.70	**83.08**	83.03	**83.08**	81.65	80.60	54.75	49.25	52.55	52.80	1.75	1.43
505	0.47	57.01	**57.02**	57.01	55.60	54.60	37.80	33.40	35.70	35.80	2.55	1.42
522	0.92	119.46	119.40	**119.52**	117.70	116.75	78.95	71.80	75.30	76.25	1.55	1.82
538	0.06	**9.01**	9.00	**9.01**	8.45	7.55	6.35	5.35	5.90	6.40	6.63	0.56
555	0.51	66.93	66.93	**66.94**	65.55	64.60	44.40	39.00	41.75	41.70	2.12	1.39
571	0.29	39.98	39.99	**40.01**	38.60	37.80	26.65	22.50	24.85	25.20	3.65	1.41
588	0.74	102.81	102.74	**102.88**	101.35	100.35	67.95	61.50	64.95	65.00	1.51	1.53
604	0.18	**27.00**	**27.00**	**27.00**	25.45	24.70	17.95	15.20	16.75	17.25	6.09	1.55
621	0.63	90.76	**90.78**	**90.78**	89.15	88.05	59.50	53.70	56.80	56.90	1.83	1.63
637	0.40	58.55	58.63	**58.68**	57.20	56.25	38.70	33.90	36.75	36.75	2.59	1.48
654	0.85	131.40	131.40	**131.46**	129.65	128.65	86.15	78.45	82.80	82.80	1.40	1.81
670	0.12	**20.32**	20.30	20.29	19.30	18.50	13.75	11.65	12.90	13.40	5.28	1.02
687	0.57	88.62	88.58	**88.72**	87.15	86.15	58.15	52.05	55.45	55.65	1.80	1.57
703	0.35	**56.00**	**56.00**	**56.00**	54.90	53.95	36.90	32.10	35.00	35.10	2.00	1.10
720	0.80	131.72	**131.81**	131.79	130.10	129.10	87.10	79.45	83.30	83.15	1.31	1.71
736	0.23	39.84	39.84	**39.91**	38.30	37.40	26.25	22.60	24.70	25.00	4.20	1.61
753	0.68	**115.03**	115.00	**115.03**	113.70	112.70	75.30	68.15	71.90	71.60	1.17	1.33
769	0.46	78.20	78.19	**78.21**	77.10	76.10	51.70	45.60	48.70	49.15	1.44	1.11
786	0.91	166.69	166.62	**166.70**	164.65	163.70	108.20	99.35	103.95	104.70	1.25	2.05
802	0.09	**18.18**	**18.18**	**18.18**	17.10	16.30	12.50	10.40	11.75	11.90	6.32	1.08
819	0.54	96.76	96.79	**96.80**	95.30	94.25	63.45	56.70	60.05	60.55	1.57	1.50
835	0.32	59.24	59.24	**59.29**	58.20	57.30	39.25	34.10	36.85	37.25	1.87	1.09
852	0.77	145.72	145.76	**145.78**	144.15	143.10	95.55	86.55	91.70	91.80	1.13	1.63
868	0.20	**40.00**	39.98	**40.00**	38.80	37.80	26.55	22.70	25.00	25.30	3.09	1.20
885	0.65	124.74	124.76	**124.81**	123.20	122.20	81.50	73.60	78.60	78.65	1.31	1.61
901	0.43	83.45	83.46	**83.53**	82.10	81.10	55.00	48.40	52.05	51.80	1.74	1.43
918	0.88	179.98	**180.01**	**180.01**	178.25	177.20	118.05	108.70	113.60	113.60	0.99	1.76
934	0.15	32.87	32.84	**32.90**	31.40	30.50	21.60	18.50	20.50	21.00	4.78	1.50
951	0.60	121.89	121.89	**121.92**	120.30	119.30	79.20	71.65	75.85	76.10	1.35	1.62
967	0.37	**76.61**	76.55	76.53	75.15	74.30	50.25	44.30	47.45	47.50	1.94	1.46
984	0.82	**175.10**	**175.10**	**175.10**	173.50	172.45	115.10	105.20	110.50	110.15	0.92	1.60
1000	0.26	56.58	**56.70**	56.61	55.60	54.60	37.35	32.30	35.20	35.80	1.98	1.10

Table 9.9: Results obtained by the solution of three ILP models.

			ILP			ILP-SM$_1$			ILP-SM$_2$		
	$\|V\|$	D	obj	time(s)	opt	obj	time(s)	opt	obj	time(s)	opt
RG	27	0.50	**8.00**	38	1.00	**8.00**	2205	0.40	**8.00**	347	0.95
	43	0.27	**6.45**	519	0.90	**6.45**	731	0.80	**6.50**	856	0.80
	60	0.73	**23.00**	3600	0.00	22.75	3600	0.00	22.85	3600	0.00
	76	0.16	**5.85**	1	1.00	**5.85**	1	1.00	**5.85**	1	1.00
	93	0.61	26.20	3600	0.00	24.15	3600	0.00	23.90	3600	0.00
	109	0.39	18.40	3600	0.00	16.75	3600	0.00	16.55	3600	0.00
	126	0.84	41.40	3600	0.00	13.95	3600	0.00	0.45	3600	0.00
	142	0.11	**7.20**	723	0.80	**7.10**	724	0.80	**7.20**	727	0.80
	159	0.56	1.00	3600	0.00	3.65	3600	0.00	0.00	3600	0.00
	175	0.33	16.80	3600	0.00	19.30	3600	0.00	8.40	3600	0.00
	192	0.78	31.80	3600	0.00	3.25	3600	0.00	0.00	3600	0.00
	208	0.22	11.40	3600	0.00	14.85	3600	0.00	10.80	3600	0.00
	225	0.67	1.00	3600	0.00	0.05	3600	0.00	0.00	3600	0.00
	241	0.44	1.00	3600	0.00	2.45	3600	0.00	0.00	3600	0.00
	258	0.89	0.00	3600	0.00	0.00	3600	0.00	0.00	3600	0.00
	274	0.08	9.10	2724	0.25	8.80	2733	0.25	8.75	2980	0.20
	291	0.53	0.10	3600	0.00	0.00	3600	0.00	0.00	3600	0.00
WS	27	0.52	**9.00**	168	1.00	**9.00**	2956	0.25	**9.00**	369	1.00
	43	0.28	**7.85**	1478	0.70	**7.85**	1951	0.50	**7.85**	1881	0.50
	60	0.73	**24.20**	3600	0.00	23.75	3600	0.00	24.00	3600	0.00
	76	0.16	**7.95**	1642	0.55	**7.80**	1823	0.50	**7.95**	1637	0.55
	93	0.60	27.00	3600	0.00	24.30	3600	0.00	24.35	3600	0.00
	109	0.39	19.90	3600	0.00	17.60	3600	0.00	17.75	3600	0.00
	126	0.84	58.80	3600	0.00	38.85	3600	0.00	0.75	3600	0.00
	142	0.11	**9.00**	3241	0.10	**8.75**	3241	0.10	**8.85**	3242	0.10
	159	0.57	1.00	3600	0.00	4.05	3600	0.00	0.00	3600	0.00
	175	0.33	15.20	3600	0.00	20.25	3600	0.00	1.00	3600	0.00
	192	0.78	31.35	3600	0.00	6.65	3600	0.00	0.00	3600	0.00
	208	0.22	12.95	3600	0.00	17.30	3600	0.00	8.50	3600	0.00
	225	0.67	1.00	3600	0.00	0.05	3600	0.00	0.00	3600	0.00
	241	0.44	1.00	3600	0.00	1.35	3600	0.00	0.00	3600	0.00
	258	0.89	0.00	3600	0.00	0.00	3600	0.00	0.00	3600	0.00
	274	0.08	9.25	3600	0.00	9.40	3600	0.00	9.40	3600	0.00
	291	0.53	0.00	3600	0.00	0.00	3600	0.00	0.00	3600	0.00
BA	27	0.52	**7.00**	1	1	**7.00**	37	1.00	**7.00**	1	1.00
	43	0.28	**6.00**	1	1	**6.00**	459	0.90	**6.00**	14	1.00
	60	0.73	17.00	3600	0	16.80	3600	0.00	17.00	3600	0.00
	76	0.16	**6.00**	20	1	**6.00**	1219	0.85	**6.00**	249	1.00
	93	0.60	19.00	3600	0	17.95	3600	0.00	18.80	3600	0.00
	109	0.39	**14.10**	3600	0	13.05	3600	0.00	13.85	3600	0.00
	126	0.84	32.55	3600	0	29.65	3600	0.00	29.35	3600	0.00
	142	0.11	**6.95**	3600	0	**6.75**	3600	0.00	**6.90**	3600	0.00
	159	0.57	22.40	3600	0	22.35	3600	0.00	8.10	3600	0.00
	175	0.33	17.20	3600	0	14.10	3600	0.00	15.65	3600	0.00
	192	0.78	1.00	3600	0	13.90	3600	0.00	0.00	3600	0.00
	208	0.22	13.90	3600	0	12.30	3600	0.00	12.65	3600	0.00
	225	0.67	1.00	3600	0	2.00	3600	0.00	0.00	3600	0.00
	241	0.44	1.25	3600	0	3.00	3600	0.00	0.00	3600	0.00
	258	0.89	1.00	3600	0	0.70	3600	0.00	0.00	3600	0.00
	274	0.08	**8.05**	3600	0	**7.05**	3600	0.00	**7.05**	3600	0.00
	291	0.53	1.00	3600	0	1.80	3600	0.00	0.00	3600	0.00

Table 9.10: Performance of the algorithms and the ILP models for different types of networks, in terms of the number of instance groups for which the respective algorithms obtain the best average solution.

	RG	WS	BA	Sum	% best
CMSA	23	30	21	74	41.1 %
CMSA-L_{ALL}	18	14	12	44	24.4 %
CMSA-$L_{1,2}$	34	39	43	116	64.4 %
CMSA (any)	58	57	55	170	94.4 %
ILP	5	5	6	16	8.8 %
ILP-SM_1	4	4	5	13	7.2 %
ILP-SM_2	4	4	5	13	7.2 %
MDDS-GH	0	0	0	0	0.0 %
IAM	0	0	0	0	0.0 %
P-MAX	0	0	0	0	0.0 %
P-MIN	0	0	0	0	0.0 %
R-LID	0	0	0	0	0.0 %
COLOR-DDS	0	0	0	0	0.0 %

Part IV

Reinforcement Learning

Chapter 10

Reinforcement Learning for Multi-Neighborhood Search and Multi-Constructor CMSA

Machine learning applied to adaptive tuning of parameters in metaheuristics is a growing research field (Adriaensen et al., 2022; Talbi, 2021). This is motivated by the fact that offline tuning remains a time-consuming and somewhat ineffective activity, despite the availability of many statistically-principled black-box automatic configuration tools (see Huang et al., 2020).

Indeed, tuning procedures struggle to find a single best configuration when instances differ considerably, either in size or in problem-specific features (Schneider and Hoos, 2012). Some alternative approaches to face scenarios with heterogeneous instances are instance clustering (Kadioglu et al., 2010), instance space analysis (Smith-Miles and Muñoz, 2023) and feature-based tuning (Bellio et al., 2016). However, they are, in general, time-consuming and may require in-depth problem-specific knowledge, and so the need for best practices is still felt as a priority by the optimization community (Bartz-Beielstein and et al., 2020).

On the contrary, online parameter control reduces the time spent for the tuning phase and is suitable for scenarios in which there is no parameter configuration that fits all instances or some validation instances end up differing substantially from the training ones. In addition, it also helps in cases in which the best parameter configuration changes, within a single run, in different stages of the search procedure and in different areas of the search space.

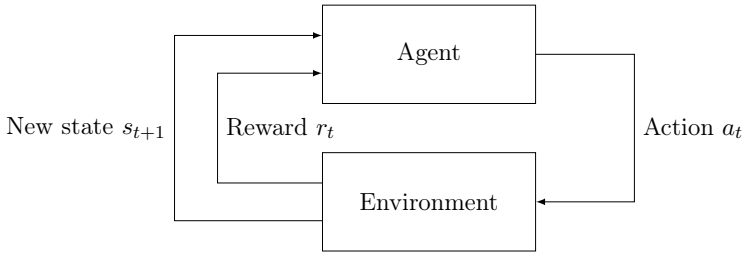


Figure 10.1: Reinforcement learning

For these reasons, we investigate the application of reinforcement learning (RL) methods to adjust the operator weights in metaheuristic search. Reinforcement learning is a learning paradigm where an intelligent agent learns the best policy for its actions by interacting with the environment. The agent takes actions in order to maximise its notion of *cumulative reward*. Fig. 10.1 shows a simple representation of the mechanism.

In particular, we are interested in using this method for the neighborhood probabilities in Multi-Neighborhood Search and for the constructor probabilities in Multi-Constructor CMSA. In Multi-Neighborhood Search, it is used for updating the rates of the neighborhoods after every temperature, with a reward function that considers both the relative improvement in objective function and the computational cost of neighborhoods. In CMSA, after the completion of every iteration, the algorithm assigns rewards to the constructors according to their relative contributions to the solution found by the exact solver at the given iteration.

Regarding Multi-Neighborhood Simulated Annealing, we test our approach on two real-world optimization problems, namely Examination Timetabling (Carter et al., 1996) and Sports Timetabling Chapter 5. We selected these problems because multi-neighborhood Simulated Annealing has already obtained state-of-the-art results upon the corresponding benchmarking datasets. Therefore, our aim is not only to improve on the version without learning, but also to obtain truly competitive results. Concerning Multi-Constructor CMSA, the only application so far is on the Maximum Disjoint Dominating Sets Problem, described in Chapter 9.

10.1 Related work

The literature in the field of parameter control is extremely vast. We refer to the comprehensive and up-to-date work of Adriaensen et al. (2022) for a review of the area, and to Doerr and Doerr (2020) for theoretical results about its effectiveness.

The interaction between machine learning techniques and optimization methods for solving hard combinatorial optimization problems has been a hot topic for both the operations research and artificial intelligence communities in recent years (Bengio et al., 2021; Karimi-Mamaghan et al., 2022; Song et al., 2019; Talbi, 2021). In particular, Karimi-Mamaghan et al. (2022) reviewed the recent contributions in the integration of machine learning into metaheuristics and proposed a taxonomy to classify the different types of integration.

To the best of our knowledge, the only previous works that employ RL techniques for the neighborhood selection within SA have been presented by Mosadegh et al. (2020) and Shahmardan and Sajadieh (2020).

Mosadegh et al. integrated a *Q-learning* algorithm into a hyper SA framework to learn the most suitable action through the search. They identified 16 admissible actions, each one composed by a triple of move operators to be executed in sequence. The reward is computed as the difference between the value of the objective function before and after applying the three operators.

Shahmardan and Sajadieh experimented several RL-based selection mechanisms, including Q-learning, all updating the credits at each iteration of the search process. A unit reward is assigned to a move if the value of the objective function is equal to or better than the current one. In order to guarantee a good balance between exploitation and exploration, they adopted a ε-greedy selection policy, such that any operator is randomly selected with probability ε, while the operator with the maximum credit is selected with probability $(1 - \varepsilon)$. The methods have been evaluated on benchmarks of an assembly line sequencing problem (Mosadegh et al., 2020) and a truck scheduling problem (Shahmardan and Sajadieh, 2020).

Other examples of integration of RL methods into other metaheuristics for the purpose of operator selection can be found in Alicastro et al. (2021); Fialho et al. (2010); Gunawan et al. (2018); dos Santos et al. (2014); Toffolo et al. (2018), with applications to Evolutionary Algorithms, Variable Neighborhood Search, Late Acceptance Hill-Climbing, and Iterated Local Search.

The use of feedback about the performance of neighborhood operators

for dynamically adjusting the probabilities in the different stages of the search process is also referred to as Adaptive Neighborhood Search (Li et al., 2015; Lü and Hao, 2012). It has found wide application in particular in combination with the Large Neighborhood Search paradigm, resulting in the Adaptive Large Neighborhood Search metaheuristic (ALNS) (Mara et al., 2022; Pisinger and Ropke, 2019), where the rates of destroying and repairing operators are dynamically adjusted using the recorded performance of the neighborhoods.

Lastly, the selection of the appropriate operator during the search process was also studied in the context of *hyper-heuristics* (Burke et al., 2019; Kheiri et al., 2021; Mischek and Musliu, 2022) In this context, the operator represents a low-level heuristic , instead of a single move. Recently, Kallestad et al. (2023) proposed a selection hyper-heuristic framework that integrates Deep RL into ALNS.

10.2 Reinforcement learning strategy

Among the possible applications of RL, we consider the *exponential recency-weighted average bandit algorithm* (see Sutton and Barto, 2018), which follow the rule at Eq. (10.1), where σ_{it} and r_{it} are the rate and the *reward* of the i-th operator at the learning iteration t, respectively, and $\lambda \in [0, 1]$ is the learning rate.

$$\sigma_{i,t+1} = (1 - \lambda)\sigma_{it} + \lambda r_{it} \tag{10.1}$$

According to the classification of Karimi-Mamaghan et al. (2022), our method implements a *low-level* integration of RL into metaheuristics, with the purpose of *operator selection* during the search evolution based on a credit assigned to each operator from its historical performance.

The parameter λ sets the pace of the learning process. As shown in Eq. (10.2), the first component is the memory of past rewards, or, in other terms, the accumulated experience from the search, and it is weighted $1 - \lambda$. The second component is the reward obtained in the last iteration, which is weighted λ. If $\lambda = 1$, there is no memory in the process, and the next probabilities are influenced exclusively by the last rewards. On the opposite, if $\lambda = 0$, no learning is involved, and the probabilities of the operators are never changed. Therefore, a proper tuning of λ is quite critical for the effectiveness of the learning mechanism.

$$\sigma_{i,t+1} = \underbrace{(1-\lambda)\cdot\sigma_{i,t}}_{\text{memory of past rewards}} + \underbrace{\lambda\cdot r_{i,t}}_{\text{learning from last rewards}} \tag{10.2}$$

If the learning iteration and the algorithm iteration do not coincide, but instead the operator weights are updated only with a given frequency, we say that the learning approach is operated in batch mode. The *learning batch* is the length of the learning iteration. There are several good reasons to learn in batches. For example, the information needed to compute the rewards might not be available at every iteration. Second, the learning mechanism is sensitive to the noise of the reward function, and learning in batches provides against the effect of outliers. Third, the computational overhead of learning at every iteration might be not negligible.

A desirable property of our learning process is that it ensures $\sum_i^K \sigma_{i,t} = 1$, at every learning iteration t, where K is the total number of operators. If the learning process starts with $\sum_i^K \sigma_{i,1} = 1$, that is, the sum of the operator rates at the beginning is 1, in order to guarantee the conservation of the sum of the probabilities it is enough to ensure that $\sum_i^K r_{i,t} = 1$. Then, at any generic iteration t we have that:

$$\sum_i^K \sigma_{i,t+1} = \sum_i^K (1-\lambda)\cdot\sigma_{i,t} + \lambda\cdot r_{i,t} = \tag{10.3a}$$

$$(1-\lambda)\cdot\sum_i^K \sigma_{i,t} + \lambda\cdot\sum_i^K r_{i,t} = \tag{10.3b}$$

$$(1-\lambda)\cdot 1 + \lambda\cdot 1 = 1 \tag{10.3c}$$

For the initial iteration, a reasonable choice is to give all the neighborhood the same probability $\frac{1}{K}$, so that $\sum_i^K \sigma_{i,1} = \sum_i^K \frac{1}{K} = 1$.

A downside of the proposed learning scheme is that an operator that, even after an increasing number of consecutive iterations, has not received any reward may have its probability decreased to a very low value. This would make it nearly impossible that this operator is selected, thus making it extremely improbable that it ever gets any future reward. Especially when λ is set to high values, an operator faces the risk to be kicked out extremely fast, even though it might be useful to have it later in the search. Therefore, in order to guarantee that an operator can always be selected, we set a minimum probability σ_{min}, so that it does not disappear from the search (*Probability Matching Selection* - PMS). We do this by correcting Eq. (10.1)

into Eq. (10.4) so that $\sigma_{i,t+1}$ is equal to the maximum between the computed value and σ_{min}:

$$\sigma_{i,t+1} = \max\{\sigma_{min}, (1-\lambda) \cdot \sigma_{i,t} + \lambda \cdot r_{i,t}\} \tag{10.4}$$

In case of a correction, all other σ_* are rescaled accordingly, so that their sum remains equal to one. Please note also that the minimum threshold σ_{min} implies that the probabilities are upper-bounded by a maximum value of $1-(K-1)\sigma_{min}$.

10.3 RL for Multi-Neighborhood Search

The key element of the whole local search paradigm is indisputably the neighborhood relation, that defines the atomic movements that allow for navigating the search space, looking for better solutions. Indeed, the crucial issue of local search is how to *escape* from local minima of the search space created by the structure defined by the neighborhood relation, by balancing exploration and exploitation. In fact, all the above-mentioned techniques include some "smart" mechanisms to move away from the basin of attraction of a local minimum.

The use of a neighborhood composed by multiple atomic ones is an additional/alternative way to overcome the possibility of getting stuck, given that a local minimum of a neighborhood is not necessarily a local minimum of another one. In a multi-neighborhood approach, the decision of which neighborhood to probe at each iteration is relevant for efficiency and effectiveness of the search.

In practice, we study Multi-Neighborhood Simulated Annealing described in Chapter 2, which relies on the iterated drawing of random moves with the *two-step move selection*: first a *biased* random selection to establish which atomic neighborhood should be sampled, and then a *uniform* selection within the chosen neighborhood. The probabilities for the first selection, called *rates*, are parameters that are subject to offline tuning, along with the parameters of SA.

To select the parameters, the typical option is to use an automatic tuning tool, such as Iterated F-Race (I/F-Race) (López-Ibáñez et al., 2016) or SMAC (Hutter et al., 2011), which selects the parameters based on runs on a training dataset. I/F-Race samples a set of parameter configurations according to particular distributions, and iteratively tests them against an increasingly large set of instances, drawn uniformly from the training dataset. At each step, the Friedman's non-parametric two-way analysis of variance by

ranks is applied to select the non-inferior configurations by means of racing (Maron and Moore, 1997). Then the sampling distributions are updated to increase the probability of sampling, in the future, parameter values from the best configurations. The procedure is repeated until the computational budget, expressed in terms of total number of experiments or overall running time, is exhausted. The current version of the IRACE package (López-Ibáñez et al., 2016) implements also a "soft-restart" mechanism to avoid premature convergence and an elitist variant for preserving the best configurations found so far.

The alternative option to offline tuning is the online control of the parameters during the search discussed in this chapter. Obviously, not all parameters can be modified online, as for some of them the value is used immediately at the beginning of the search. For example, the initial temperature of SA must necessarily be fixed offline.

In our proposal, the rates σ_* are adapted online based on the RL approach presented in Section 10.2, whereas the parameters of SA are tuned offline. The learning mechanism is applied after a given batch of iterations, and not upon every single move execution. In this context, the natural option is to synchronize the learning action with the annealing step, so that the new rates are computed in the very same iteration in which the temperature is decreased. This choice limits the overhead of the learning procedure upon the search method. On the contrary, given that normally the number of executed moves in SA is extremely large, learning at any move execution would result in an unacceptable overhead.

Given that we operate in batch mode, we aggregate the performance of all the operators over an entire temperature level (*Average Credit Assignment* - ACA). To define the reward r_{it} in Eq. (10.1), we first compute the *score* φ of neighborhood N_i at temperature level t as

$$\varphi_{it} = \frac{\sum_{j}^{n_{it}} |\min\{\Delta F_j, 0\}|}{n_{it}} \tag{10.5}$$

where n_{it} is the total number of drawn moves of neighborhood N_i at temperature level t and the summation term is the contribution to cost minimization. In other words, we consider as score the total improvement generated by the neighborhood upon the previous temperature divided by the number of drawn moves, so as not to bias the reward toward neighborhoods with high rates.

Given that the computational cost for the construction and the evaluation of a move might be different from neighborhood to neighborhood, the computational effort has to be taken into account in the learning phase,

by applying a reward formula that penalizes the neighborhoods that take more time (see Mischek and Musliu, 2022).To this aim, we insert the average running time of moves of N_i at temperature level t, called τ_{it}, as denominator of the formula that computes the reward, as seen in Eq. (10.6). However, the penalization should be smoothed in order to give credit to the moves that find improvements, even if in a computationally expensive way. We perform such smoothing by using of the following exponential function

$$R_{it} = \frac{\varphi_{it}}{(\tau_{it})^m} \tag{10.6}$$

where m (with $0 < m \leq 1$) is a novel real-valued hyperparameter. To ensure that the rates add up to one, the rewards are normalized by dividing each reward by the sum of all rewards as follows:

$$r_{it} = \frac{R_{it}}{\sum_{j=1}^{K} R_{jt}} \tag{10.7}$$

This normalization guarantees that all the terms used in Eq. (10.1) add up to one and, consequently, also the updated selection rates will always sum up to one.

In conclusions, the hyperparameters of the learning procedure are the learning rate (λ), the rate threshold (σ_{min}), and the exponent m of Eq. (10.6). To these three, we add the SA cooling rate α, given that it governs not only the duration of the annealing step, but also the learning batch. The best values for these four hyperparameters are computed by the tuning procedure discussed in Section 10.3.2.

10.3.1 Case studies

We propose as case studies two real-world problems coming from the field of timetabling: examination timetabling and sports timetabling. Both come along with a challenging dataset that has been used extensively in the literature.

Examination Timetabling

Examination timetabling (ETT) is one of the problems that every university has to deal with on a regular basis. Many formulations of ETT have been proposed in the literature. We consider here the classical and essential version proposed by Carter et al. (1996), which is the most studied one (see the recent survey Ceschia et al., 2023).

The input data in this formulation is just the Boolean-valued *enrollment* matrix, that stores for each pair $\langle \text{student}, \text{exam} \rangle$ the information about whether the student has to take the exam or not. Two exams with one or more students in common are in conflict, and so they cannot be scheduled in the same period. The objective function is based on the distance between exams with students in common. Distances are penalized in the following way: the cost of scheduling two exams with k students in common at distance equal to 1, 2, 3, 4, and 5 periods is $16k$, $8k$, $4k$, $2k$, and k, respectively.

Many authors have tackled this problem by using local search in general and multi-neighborhood search in particular. We refer here to the approach by Bellio et al. (2021) that propose a two-stage SA procedure based on a comprehensive set of neighborhoods. They obtained state-of-the-art results on the Toronto dataset, which is the standard ground for comparison for ETT and has been used in many previous studies. In detail, Bellio et al. collected and implemented five different neighborhood relations from the literature. The first three neighborhoods reschedule one, two, or three exams, respectively; the fourth swaps all the exams of two periods, and the last one performs a Kempe chain. They are called MoveExam (ME), KickExam (KE), DoubleKickExam (DKE), SwapPeriods (SP), and KempeChain (KC), respectively. These neighborhoods exhibit different computational costs for constructing and evaluating the move. In particular, Bellio et al. showed experimentally that the computation cost of the construction and evaluation of a DKE or KC move is more than 30 times the cost of a ME move. For this reason, their tuning procedure which uses F-Race, takes into account the average computational cost of each neighborhood. They compare the alternative configurations by fixing the total running time of SA. Coherently, the temperature is decreased after a given allotted time, and not based on the given number of samples.

The first stage of Bellio et al. aims at obtaining the first feasible solution and it is rather fast. For this reason, we focused on the second stage. For the second stage, we use the same neighborhoods and also the same configuration for the SA parameters; by contrast, all rates σ_* are uniformly initialized to $1/K$ (i.e., 0.2 in the specific case, since $K = 5$) and are adjusted dynamically by the learning procedure.

Sports Timetabling

The problem that we consider is described in Chapter 5. We just recall that our solution method is a Multi-Neighborhood Simulated Annealing that uses a collection of six neighborhoods. Five of them are classical ones for

Sports Timetabling, namely SwapHomes (SH), SwapTeams (ST), SwapRounds (SR), PartialSwapTeams (PST) and PartialSwapRounds (PSR), plus an original contribution, named PartialSwapTeamsPhased (PSTP).

In the original work, we tuned offline the probabilities of the neighborhoods, assigning a unique value for the three stages, but differentiated between phased and not phased instances. In this context, we employ the same configuration for the SA parameters, but instead of fixed neighborhood rates we implement the learning algorithm described in Section 10.3, starting every stage with equal probabilities of 1/6 for every neighborhood (given that $K = 6$).

10.3.2 Experimental results

Our search methods are implemented in C++, as both the code of Bellio et al. and our ITC2021 solver are in C++ as well[1]. The code was compiled on Ubuntu Linux 20.4 using g++ (v. 11.3) in `-O3` mode, and the experiments were run on AMD Ryzen Threadripper PRO 3975WX 32-Cores (3.50 GHz), with one single core dedicated to each experiment.

Contrarily to the works presented in Part II, we do not set the total number of iterations $\mathcal{I}$ in advance, but we run the algorithm on a fixed runtime, that is distributed uniformly among the temperatures of the SA. This choice is motivated by the fact that the evolution of the neighborhood rates during the search may yield very different runtimes from those of Bellio et al. and Chapter 5, depending on the share of rewards taken by slower and faster neighborhoods. Therefore, we fix for every instance the same runtime used by the algorithms from Bellio et al. and Chapter 5, respectively for ETT and STT, scaled based on the ratio of performance on the different machines, so that the comparison is done on the same computational budget. This choice sacrifices the reproducibility of the individual experiments, but it is necessary to understand the contribution of the learning mechanism.

The tuning procedure was performed using I/F-race separately for the two problems on the learning hyperparameters λ, σ_{min}, m, and α. The total budget assigned for each problem was 5000 experiments, with an average running time for each experiment of 300s and 2000s for ETT and STT, respectively. Table 10.1 shows the initial ranges of the hyperparameters and the two best configurations.

For both problems, it turned out that there is statistical significance between the "winning" configuration and all the configurations with $\lambda = 0.0$,

[1]The code of Bellio et al. is available online at `https://opthub.uniud.it` and the code from Chapter 5 at `https://github.com/robertomrosati/sa4stt`.

Table 10.1: Learning hyperparameter configurations

Hyperparameter	Initial range	Best configuration ETT	Best configuration STT
λ	[0.0, 0.5]	0.038	0.093
σ_{min}	[0.0, 0.01]	0.000	0.001
m	[0.16, 1.0]	0.197	0.259
α	[0.98, 0.995]	0.987	0.983

which are consistently eliminated by the race procedure, confirming that the learning mechanism is necessary. On the contrary, there are many other configurations with $\lambda > 0.0$ that are statistically equivalent, showing that the results are robust with respect of the hyperparameter configuration.

It is worth noting that the selected value for σ_{min} is 0.0 for ETT. This means that it is actually possible that a given neighborhood becomes useless from some point onward in the search.

Results for Examination Timetabling

The results for 30 repetitions on the Toronto dataset using the configuration of Table 10.1 are shown in Table 10.2. They are granted the same running time[2] of Bellio et al. (2021), which are the best known ones among those with relatively short running time. The gaps reported in Table 10.2 are computed as the percentage difference with respect to Bellio et al..

We consider for our learning procedure two alternative settings: with uniform initial rates discussed so far (0.2 for all) and with the initial rates reported by Bellio et al. (namely 0.757, 0.144, 0.001, 0.058, and 0.04).

The outcome is that both configurations perform better than the offline tuning with no learning. Even though the improvement is quite small in absolute terms, it is actually statistically significant. This is confirmed by the Wilcoxon signed-rank test that returns a p-value close to 0 (equal to $7.192 \cdot 10^{-6}$) upon 15 repetitions on each instance, thus strongly rejecting the null hypothesis.

Between the two learning configurations, there is a minimal advantage to the uniform one, which finds the best average results (in boldface) in 9 out of 13 instances. However, using the same test there is no statistical significant between these two. Nonetheless, we believe that the reason for the slightly

[2]The running time has been scaled based on the ratio of performance on the different machines.

Table 10.2: Comparison results for ETT

		Bellio	reinforcement learning			
		et al.	uniform		tuned	
	time	(2021)	initial rates		initial rates	
Inst.	secs	avg	avg	% gap	avg	% gap
car91	688.1	**4.44**	**4.44**	0	**4.44**	0
car92	544.6	3.80	3.80	0	**3.79**	-0.263
ear83	249.6	32.89	**32.85**	-0.122	32.91	0.061
hec92	228.5	10.16	10.17	0.098	**10.15**	-0.098
kfu93	217.5	13.06	**13.03**	-0.23	13.06	0
lse91	209.3	10.09	10.03	-0.595	**10.02**	-0.694
pur93	1382	4.32	**4.29**	-0.694	4.30	-0.463
rye93	281.7	8.10	**8.07**	-0.37	**8.07**	-0.37
sta83	136.6	157.05	**157.04**	-0.006	157.05	0
tre92	296.6	7.85	**7.82**	-0.382	7.83	-0.255
uta92	575.8	3.13	**3.11**	-0.639	3.12	-0.319
ute92	130.8	**24.82**	24.83	0.04	24.83	0.04
yor83	429.6	34.93	**34.91**	-0.057	34.96	0.086
avg	413.1	24.20	**24.18**	-0.227	24.19	-0.175

better performance of the uniform initial rates come from the fact that in the beginning of the search all types of moves are useful to "shake" the initial solution. Going on, the most effective moves become prominent. This is confirmed by Fig. 10.2 that shows the evolution of the rates for one specific run of each configuration (uniform left and tuned right, for instance ear83). We see that indeed the rates tend to stay uniform or become uniform in the initial part of the search, and then diversify in the final part.

Results for Sports Timetabling

The results for 15 repetitions on the ITC2021 dataset using the configuration of Table 10.1 are shown in Table 10.3, with the same (normalized) running time of Chapter 5. The column "feas" reports the rate of feasible solutions found.

Our approach yields better results (in boldface) for 27 out of 45 instance, and equal value for 5. The average percentage improvement is 0.54 with a high variance, having differences that go from -16.53 to 45.32. The

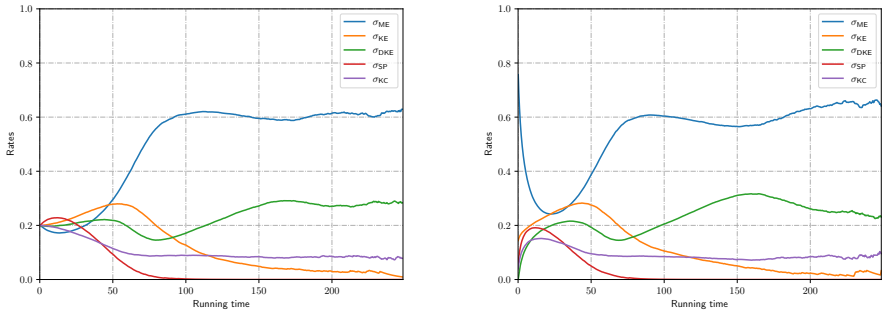


Figure 10.2: Evolution of σ_* for ETT (instance ear83) for uniform and tuned initial rates

improvement is statistically supported by the Wilcoxon signed-rank test with a p-value equal to 0.03, even if less significantly than for ETT. It is worth mentioning that our results are outperformed only by the matheuristic approach by Lamas-Fernandez et al. (2021), which however grant much longer running times.

The evolution of the rates for two instances of STT is shown in Fig. 10.3. We notice that the behavior is rather different in the two cases, demonstrating that there is no single configuration that fits all cases.

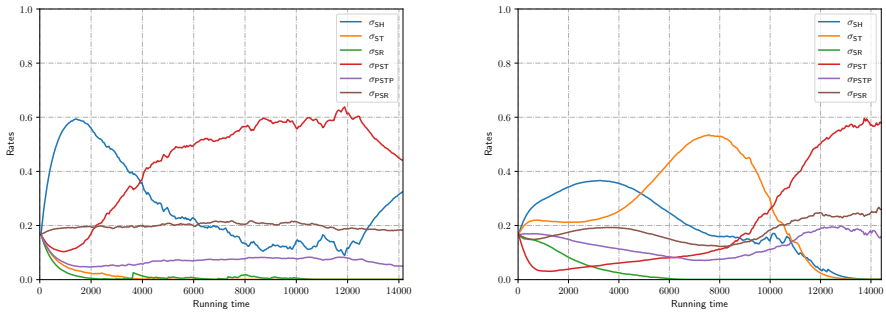


Figure 10.3: Evolution of σ_* for STT (instances Middle_10 and Middle_12, stage II)

Table 10.3: Comparison results for STT

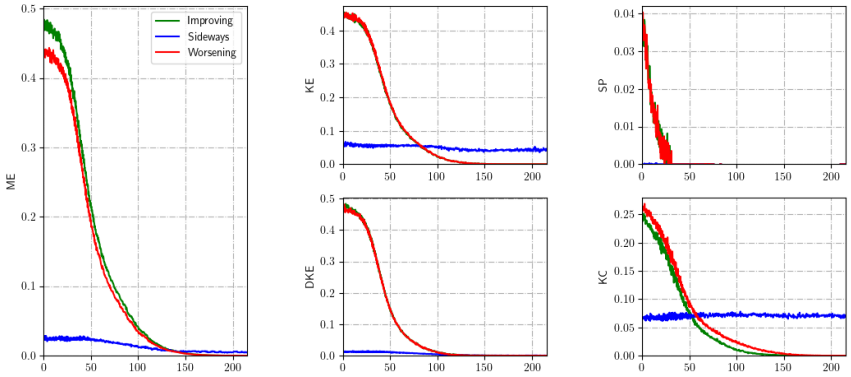
Inst.	tuned avg	tuned feas	learning avg	learning feas	% gap
E_1	540.7	1.00	**530.8**	1.00	-1.83
E_2	384.6	1.00	**375.6**	1.00	-2.34
E_3	1176.5	1.00	**1175.1**	1.00	-0.12
E_4	1007.8	0.56	**943.8**	0.56	-6.35
E_5	-	0.00	-	0.00	
E_6	4543	1.00	**4411**	1.00	-2.91
E_7	6721.7	1.00	**6263.3**	1.00	-6.82
E_8	1151.9	1.00	**1147.5**	1.00	-0.38
E_9	228.7	1.00	**190.9**	1.00	-16.53
E_10	-	0.00	-	0.00	
E_11	5784.5	1.00	**5521.8**	1.00	-4.54
E_12	1200.2	1.00	**1194.4**	1.00	-0.48
E_13	233.8	1.00	**232.2**	1.00	-0.68
E_14	**82.3**	1.00	119.6	1.00	45.32
E_15	3945.8	1.00	**3933.9**	1.00	-0.30
M_1	6075.0	0.06	**5936**	0.06	-2.29
M_2	-	0.00	-	0.00	
M_3	11403.1	0.23	**11379.7**	0.20	-0.21
M_4	33.0	1.00	**32.1**	1.00	-2.73
M_5	**624.4**	1.00	631.5	1.00	1.14
M_6	**2186.3**	1.00	2316.9	1.00	5.97
M_7	**2452.7**	1.00	2474.2	1.00	0.88
M_8	196.6	1.00	**191.1**	1.00	-2.80
M_9	772.1	1.00	**722.8**	1.00	-6.39
M_10	1687.5	1.00	**1648.9**	1.00	-2.29
M_11	2996.5	1.00	**2987.3**	1.00	-0.31
M_12	**1054.2**	1.00	1061.5	1.00	0.69
M_13	**479.3**	1.00	486.9	1.00	1.59
M_14	1304.6	1.00	**1267.6**	1.00	-2.84
M_15	1099.7	1.00	**1095.3**	1.00	-0.40
L_1	2372.7	1.00	**2327.9**	1.00	-1.89
L_2	6085.5	0.49	**6040**	0.38	-0.75
L_3	**2718.0**	1.00	2731.3	1.00	0.49
L_4	**0.0**	1.00	**0.0**	1.00	0.00
L_5	-	0.00	-	0.00	
L_6	**1121.3**	1.00	1138.6	1.00	1.54
L_7	**2226.5**	1.00	2252.1	1.00	1.15
L_8	**1155.3**	1.00	1158.5	1.00	0.28
L_9	881.2	1.00	**851.8**	1.00	-3.34
L_10	3527.3	0.05	**3407.0**	0.12	-3.41
L_11	**289.3**	1.00	291.4	1.00	0.73
L_12	4830.6	1.00	**4384.7**	1.00	-9.23
L_13	**2285.5**	1.00	2299.6	1.00	0.62
L_14	**1326.3**	1.00	1329.2	1.00	0.22
L_15	82.8	1.00	**82.4**	1.00	-0.48
avg	2152.9	0.83	**2111.4**	0.83	-0.54

10.3.3 Discussion

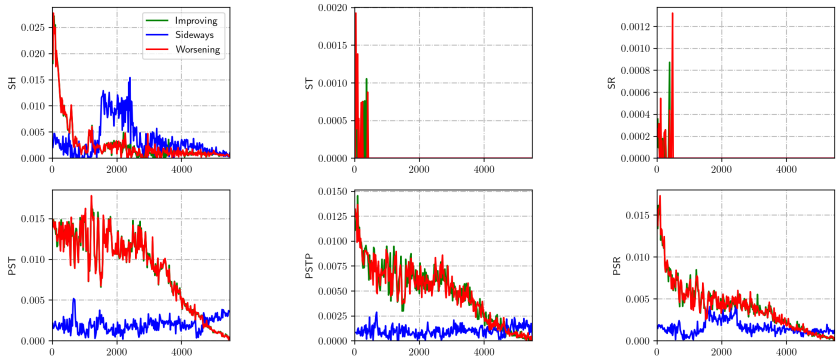
As mentioned earlier, a wide range of hyperparameter configurations can yield statistically similar outcomes. This suggests that the computational effort associated with fine-tuning the learning hyperparameters can be significantly reduced, which is typically not the case when optimizing search parameters offline.

Further validation experiments were conducted using different configurations, which confirmed the findings. For instance, we obtained the same average gap by using the optimal hyperparameter settings of STT for ETT. This indicates that the process of fine-tuning hyperparameters can be performed in a cross-domain manner, leading to significant computational savings.

Figs. 10.4a and 10.4b give some insights on the behavior of the different neighborhoods over time for one instance of ETT and STT, respectively. Specifically, they show the proportion of executed moves that are improving,



(a) Neighborhood behavior for ETT (instance kfu93)



(b) Neighborhood behavior for STT (instance Late_10)

Figure 10.4: Examples of neighborhood behavior for the two case studies

sideways, and worsening for each neighborhood, relative to the total number of evaluated moves.

The analysis reveals significant disparities in the move quality among the various neighborhoods. For example, the number of sideways moves is very different among them. Additionally, for both problems we observe a consistent trend in the number of improving and worsening moves (represented by the green and lines, respectively). This can be intuitively explained by the fact that whenever SA accepts a worsening move, there is frequently a compensating move of the same type that recovers the "damage" done.

10.4 RL for Multi-Constructor CMSA

In Chapter 9 we introduced the Multi-Constructor CMSA that selects stochastically the constructor to use at every solution generation. With K constructors, we use a vector of probabilities $(\sigma_1, \ldots, \sigma_K)$, with $\sum_{i=1}^{K} \sigma_i = 1$, where σ_i is the probability of the i-th constructor. During the construct phase of CMSA, before the generation of a new solution, one of the constructors is selected by means of a biased random selection, according to the given probabilities. Then, the next solution is built using the selected constructor.

As we have discussed previously in this chapter, the choice for the constructor rates lies between using fixed values determined by an offline tuning procedure and an online tuning strategy, based on reinforcement learning. Hereby, we compare both approaches on the Maximum Disjoint Dominating Sets Problem, described in Chapter 9, the only application, to date, of the Multi-Constructor CMSA.

While the functioning of the offline tuning approach is straightforward, hereafter we explain the characterizing aspects of the online tuning approach. First, the probabilities are initialized to $1/K$ at the first CMSA iteration. Then, at every iteration, the constructors receive a reward and their probabilities are updated depending on the relative rewards. The learning scheme is the one shown in Eq. (10.4), based on the update of the rates depending on a reward function $r_{i,t}$ and a learning rate λ. It also ensures that in any case the rates do not go below the minimum probability threshold σ_{min}.

A critical design choice is that of the reward function. The quality of the solutions generated by the constructors alone might not be a good predictor of their usefulness inside CMSA and a better metric is needed. Therefore, we propose to reward the constructors proportionally to the number of generated solution components that are found in the solution $\mathcal{S}_{\text{exc}}$ obtained by the exact solver. The following Eq. (10.8) shows the formula for assigning the rewards in the MDDSP:

$$r_{i,t} = \frac{|\mathcal{S}_{\text{exc}} \bigcap \mathcal{C}'_i|}{|\mathcal{S}_{\text{exc}} \bigcap \mathcal{C}' \setminus \mathcal{C}'_r|} \tag{10.8}$$

where $\mathcal{C}'_i$ is the set of the dominating sets in solution $\mathcal{S}_{\text{exc}}$ that have been found by the i-th constructor. To guarantee that all components give the same contribution to the reward function and also that $\sum_i^K r_{i,t} = 1$, the reward related to components found by multiple constructors is divided by the number of constructors that generated the component. The formula accounts also for the dominating sets generated heuristically by the repair

procedure that are possibly found in $\mathcal{S}_{\text{exc}}$, that are removed from the count. They are grouped in the set $\mathcal{C}'_r$.

The learning process is synchronized with the CMSA iteration. This choice is quite natural because our reward function, for evaluating the quality of different constructors, can only be calculated after the SOLVE phase of each iteration. Therefore, the *learning batch* coincides with the CMSA iteration, and the probabilities are updated every n_{sols} solution constructions.

Results

Table 10.4: Tuning values for the learning hyperparameters of CMSA-L_{all} and CMSA-$L_{1,2}$, and constructor rates in CMSA-L_{all}.

Parameter	Domain	CMSA-$L_{1,2}$	CMSA-L_{ALL}	
			learning	tuning
λ	[0.00, 1.00]	0.30	0.98	–
σ_{min}	[0.00, 0.10]	0.08	0.04	–
$\sigma_{\text{MDDS-GH}_r}$	[0.00, 1.00]	–	–	0.543
σ_{IAM_r}	[0.00, 1.00]	–	–	0.102
$\sigma_{\text{P-MAX}_r}$	[0.00, 1.00]	–	–	0.056
$\sigma_{\text{P-MIN}_r}$	[0.00, 1.00]	–	–	0.102
$\sigma_{\text{R-LID}_r}$	[0.00, 1.00]	–	–	0.095
$\sigma_{\text{COLOR-DDS}_r}$	[0.00, 1.00]	–	–	0.102

Table 10.4 shows the results of the hyperparameter tuning in CMSA-$L_{1,2}$ and CMSA-L_{all}. For CMSA-L_{all}, it also shows the results of the tuning of the fixed probabilities of the neighborhoods. Regarding the learning hyperparameters, in the case of CMSA-L_{all} a very high value is obtained for λ. This indicates that it can already be deduced from the first learning batch which constructors deserve higher weights. By the way, the final value of 0.04 for σ_{min} indicates that a certain grade of exploration is required anyway. On the other hand, CMSA-$L_{1,2}$ requires a setting of $\lambda = 0.30$ and $\sigma_{min} = 0.08$, which implies a more moderate learning pace and a higher minimum threshold: both constructors are producing useful solution components. Regarding the tuning of fixed probabilities, MDDS-GH$_r$ gets the highest weight, indicating that it is the most useful among the constructors, while the others get smaller rates, spanning from 0.056 to 0.102. We only test the case of fixed probabilities on CMSA-L_{all}, which uses all constructors.

Table 10.5: Results on random graphs: learning vs tuning in CMSA-L$_{\text{ALL}}$

Instance		CMSA-L$_{\text{ALL}}$		Instance		CMSA-L$_{\text{ALL}}$	
$\|V\|$	D	learning	tuned	$\|V\|$	D	learning	tuned
27	0.50	7.96	7.96	522	0.92	260.74	260.74
43	0.27	**6.26**	6.21	538	0.06	**12.97**	12.94
60	0.73	23.00	23.00	555	0.51	**95.84**	95.81
76	0.16	5.70	5.70	571	0.29	**56.10**	56.05
93	0.61	**28.72**	28.65	588	0.74	**157.96**	157.95
109	0.39	19.13	**19.14**	604	0.18	**38.03**	38.01
126	0.84	60.02	**60.07**	621	0.63	137.06	137.06
142	0.11	**6.80**	6.79	637	0.40	**84.16**	84.15
159	0.56	38.83	**38.90**	654	0.85	217.99	217.99
175	0.33	**23.77**	23.74	670	0.12	29.00	29.00
192	0.78	64.10	64.10	687	0.57	**132.84**	132.80
208	0.22	**18.82**	18.80	703	0.35	79.94	79.94
225	0.67	**65.54**	65.42	720	0.80	236.78	**237.93**
241	0.44	**41.20**	41.16	736	0.23	55.98	55.98
258	0.89	128.30	**128.31**	753	0.68	**182.72**	182.66
274	0.08	**9.11**	9.05	769	0.46	**113.32**	113.25
291	0.53	**57.76**	57.73	786	0.91	322.62	**323.93**
307	0.30	**34.54**	34.45	802	0.09	**26.03**	26.01
324	0.75	107.86	**107.90**	819	0.54	**141.43**	141.40
340	0.19	**24.94**	24.93	835	0.32	**84.63**	84.54
357	0.64	87.84	**87.96**	852	0.77	**232.53**	232.44
373	0.42	**56.76**	56.74	868	0.20	**56.56**	56.49
390	0.87	147.96	**148.16**	885	0.65	**189.38**	189.31
406	0.13	**20.63**	20.54	901	0.43	**121.50**	121.44
423	0.58	**87.42**	87.39	918	0.88	306.00	306.00
439	0.36	**55.44**	55.38	934	0.15	**46.72**	46.66
456	0.81	152.00	152.00	951	0.60	185.92	185.92
472	0.25	**41.94**	41.89	967	0.37	110.22	**110.25**
489	0.70	**122.59**	122.54	984	0.82	323.32	**324.64**
505	0.47	81.11	**81.12**	1000	0.26	**80.78**	80.75

The results of the comparison between CMSA-L$_{\text{ALL}}$ that employs the learning approach and the corresponding algorithm with tuned constructor rates on random graphs is shown in Table 10.5. The results reported in the case of adaptive tuning are taken from Chapter 9, while the results with fixed rates are obtained under the same experimental setting and the same repetitions per instance. Analogously to what we observe in the case of Multi-Neighborhood Search, there is no absolute "winner". However, the learning approach gets better results than the fixed probabilities on most instances groups, 35 out of 60. The remaining groups resulted in 13 ties and only 12 wins for the tuned probabilities. Clear winners are shown in boldface

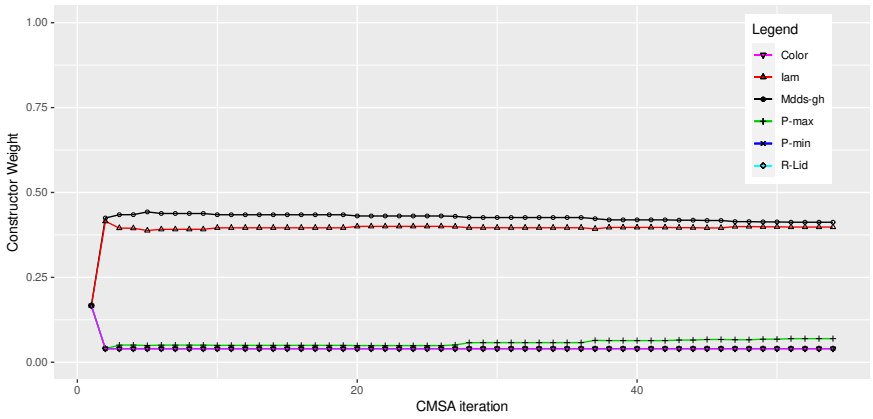


Figure 10.5: Evolution of the probabilities of the six constructors during a run of CMSA-L$_{\text{ALL}}$, for a random graph from the group ($|V| = 126$, $D = 0.84$).

in Table 10.5.

Figs. 10.5 and 10.6 display examples of the evolution of the constructor probabilities, obtained from a run of CMSA-L$_{\text{all}}$, and from two distinct runs of CMSA-L$_{1,2}$. In Fig. 10.5 we can observe the effect of having a high learning rate, set to 0.98. The probabilities converge very sharply toward their final values, starting already in the first iteration. The two constructors MDDS-GH$_r$ and IAM$_r$ receive almost all the reward and the weights of the other constructors are decreased to the minimum threshold $\sigma_{min} = 0.04$. The fact that IAM$_r$, that was assigned a quite low rate from the offline tuning procedure, converges practically to the same probability of MDDS-GH$_r$, shows the capability of the online tuning approach to adapt to the characteristic of specific instance. Having said that, it can be observed that also the constructor P-MAX$_r$ receives some reward in the final phases of the search, which increases its relative weight. Fig. 10.6 contains two patterns of the evolution of the probabilities concerning CMSA-L$_{1,2}$, from different runs on the same graph. Given that there are only two constructors and the probabilities keep summing to 1, the graphic is symmetric. In the two examples in Fig. 10.6, the probabilities of the two constructors converge toward similar values, although through a different evolution, especially at the beginning of the search process.

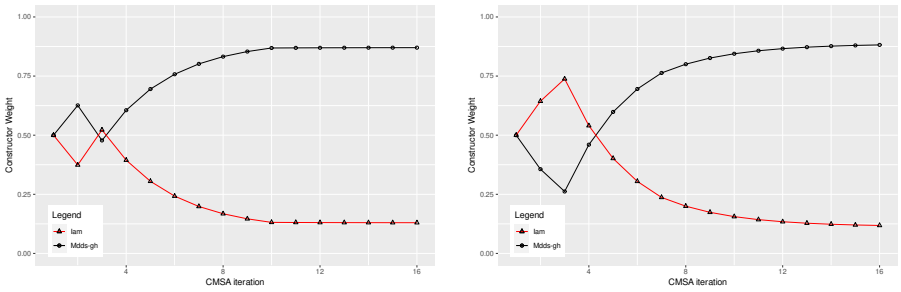


Figure 10.6: Two examples of the evolution of the probabilities of the two constructors during runs of CMSA-L$_{12}$, for an instance from the group ($|V| = 934$, $D = 0.15$).

10.4.1 Conclusions

We proposed a RL approach for Multi-Neighborhood Simulated Annealing and for the Multi-Constructor CMSA.

In Simulated Annealing, the learning procedure is not activated at each iteration, but along with the cooling step, so that its overhead is absolutely minimal and negligible with respect to the computational cost of drawing the moves and computing their costs.

The experimental results of both case studies demonstrate an improvement over already competitive baselines. In addition, our approach shows high robustness towards the configuration of the learning hyperparameters. Therefore, the application of reinforcement learning for adaptive tuning of parameters in multi-neighborhood search appears to be a promising direction for future research. Notably, the results on ETT improve over a very large body of research that has already been conducted on the challenging Toronto benchmark dataset.

In the Multi-Constructor CMSA we employ an analogous learning scheme, that rewards the constructors that provide the most useful solution components to the exact solver. The only application is on the MDDSP, in the variants that use two and six constructors. In the case of six constructors, we performed a comparison between the learning approach and the tuning of fixed probabilities, with the outcome that the learning approach obtains better results than the fixed probabilities on most instance groups.

Chapter 11

Conclusions

This thesis focused primarily on Multi-Neighborhood Search, which is a local search paradigm based on the composition of multiple neighborhoods. Compared with the single neighborhood, it provides a better connectivity in the search space and gives access to different search patterns, while also reducing the risk of getting stuck in a particular region of the search space.

The others research directions that we discussed are the Multi-Constructor CMSA, that is based on the composition of multiple constructors, and the application of reinforcement learning to both Multi-Neighborhood Search and Multi-Constructor CMSA for the online tuning of operator weights.

11.1 Research contributions

We presented, in Chapter 2, a methodological approach to design Multi-Neighborhood Search methods under a stochastic framework. Neighborhoods are associated with probabilities and moves are selected with a two-step procedure: first, a neighborhood is selected according to the probabilities; then, a move is selected uniformly within the neighborhood.

We considered all aspects involved in the local search procedure, such as the search space, the cost function, the acceptance criterion, and the metaheuristic that guides the search. We considered also peculiar aspects of MNS, including the exploration criterion, the use of internal biases, the differential cost evaluations, and possible compositions of neighborhoods. We considered our approach in integration with Simulated Annealing, which is suitable for the stochastic exploration of the neighborhoods.

Our work on MNS does not concern solely the combination of

neighborhoods together, but we actually conceived and engineered novel local search neighborhoods for specific problems. In certain case, the novel neighborhood are confirmed by statistical tests to be key components of the success of the algorithm.

Extending the idea of composing multiple operators studied in the context of MNS, we propose the Multi-Constructor CMSA, that is described in Chapter 3. It uses a probabilistic approach to select the constructor to be applied at each iteration.

Finally, in Chapter 10 we discussed the application of reinforcement learning to both Multi-Neighborhood Search and Multi-Constructor CMSA for the online tuning of operator weights.

Other contributions include the release of new instance datasets, together with instance and solution validators, and, in specific cases, of our code.

11.2 Results

To assess the validity of our MNS approach, we experimented it on four distinct combinatorial problems arising from various domains. The first one is the Minimum Interference Frequency Assignment Problem described in Chapter 4, where we employed a Multi-Neighborhood Simulated Annealing that makes use of a portfolio of six neighborhoods, two of them managed through internal biases. Chapter 5 is dedicated to the Sports Timetabling Problem, in the formulation that emerged from the 5th International Timetabling Competition ITC2021. We make use of a combination of six neighborhoods, exploited by a three-stage SA. One of them, PartialSwapTeamsPhased, was specifically designed to deal with the phased formulation. The third application is the Home Healthcare Routing and Scheduling Problem discussed in Chapter 6, that we solve with a Multi-Neighborhood Search approach that makes use of three neighborhoods. Finally, in Chapter 7 we propose a Multi-Neighborhood Simulated Annealing for the Capacitated Dispersion Problem, that uses three neighborhoods adapted from analogous proposals in the literature, with internal biases.

Regarding CMSA, we solved a Bus Driver Scheduling Problem with complex break constraints, in Chapter 8, while we applied the novel Multi-Constructor CMSA to the Maximum Disjoint Dominating Sets Problem in Chapter 9. It adapts six greedy algorithms from the literature as constructors.

We show that both our MNS and our CMSA are robust and competitive solution methods for the problems that we have considered. Properly tuned,

they quite consistently outperform the state-of-the-art from the literature. For what concerns, in particular, MSN, we compare favorably also with many other local search approaches and with more complex metaheuristics, demonstrating that our proposed method is both simple and effective. We also showed for both MNS and CMSA that the search methods that use reinforcement learning for the adaptive rates are superior in terms of solution quality with respect to the analogous versions that use fixed operator rates.

11.3 Future research directions

There are many interesting research directions that emerge as a possible continuation of this thesis.

First of all, regarding MNS, we plan to evaluate the Multi-Neighborhood Search approach within other metaheuristic frameworks, such as Tabu Search, in order to understand the impact of the search strategy on the effectiveness and the performance of the method. We also plan to investigate other ways to compose neighborhoods, such as the Cartesian neighborhood presented at the end of Chapter 2.

For the problems discussed in Chapters 4 to 7, future work will be devoted to the design of novel neighborhoods that can further increase the connectivity in the search space and the performance of the multi-neighborhood. In the case of MIFAP (Chapter 4), we consider the design of large neighborhoods based on the concept of *cell-reoptimization*, reassigning simultaneously all transmitters of a single cell. In the case of STT (Chapter 5), a promising direction is the design of new neighborhoods specifically aimed at reducing the number of breaks in the timetable, which might even be decisive toward the feasibility issue. For example, we consider the integration of the neighborhood proposed by Januario and Urrutia (2016) in our multi-neighborhood. In the case of HHCRSP (Chapter 6), we plan to design neighborhoods that can be efficient also in tackling other formulations of the problem, such as the multi-modal formulation (Rendl et al., 2012), that involves the additional choice of the transportation mode. Finally, in the case of CDP (Chapter 7), we plan to extend and refine our neighborhoods including also the ones proposed by Mladenović et al. (2022), in order to understand if this improves our results.

For all problems, we also consider studying how well our solution methods adapt to related formulations. In the case of MIFAP, we think of new formulations that will emerge from recent technologies, such as 5G wireless networks. Regarding STT, our solution method can be easily extended to

many existing variants of the problem. The task is eased by the fact that many instances for various formulation have been collected and converted into the RobinX instance format by Van Bulck et al. (2020). For HHCRSP, in addition to multi-modality, we will consider the synchronization based on mobile equipment, the presence of multiple depots, and penalties for caregiver idleness and overtime. Furthermore, we plan to address the case of the multi-day planning horizon, which brings in many new issues related to the stability of caregivers and service time for patients over the days. Regarding the CDP, our solution method can be extended to the generalized dispersion problem, or to other diversity problems.

In the case of the Multi-Constructor CMSA, we plan to employ it also for the solution of the BDS (Chapter 8). Moreover, we will devote future work to study how to integrate atomic constructors that don't necessarily generate complete solutions. In the case of BDS, this can be done by having constructors that generate individual bus shifts, while in the case of the MDDSP (Chapter 9), through constructors that generate individual dominating sets.

Besides distinct formulations of the aforementioned problems, a natural extension of this thesis is the solution of other hard combinatorial optimization problems by Multi-Neighborhood Search and CMSA. The former might yield good results on problems where local search is known to be effective, such as those arising in scheduling and timetabling. The latter can be employed to solve problems other problems in the domain of personnel rostering, like railway or airline crew scheduling. A comparison between the performances of the two methods in order to gain insights on their relative footprint is also an interesting research direction.

Moreover, the encouraging results from Chapter 10 suggest that we should investigate the systematic integration of reinforcement learning for adaptive operator weight tuning in both MNS and Multi-Constructor CMSA. The main challenge in this regard is to define a reward function that is neighborhood- and problem-independent. It would be interesting to investigate about the possible definitions of state, based on the trend of the search (e.g., plateaux, steep slope, basin of attraction of a local minimum).

Finally, an ambitious research direction is the design a Multi-Neighborhood Search approach that does not rely on a metaheuristic. Similarly to hyper-heuristics, this could be done on the basis of reinforcement learning (Drake et al., 2020; Mischek and Musliu, 2022). The multi-neighborhood would behave as an intelligent agent on the search space, guided, for example, by a *Q-learning* scheme.

List of publications

We present hereafter the list of the publications related to this thesis. We only list publications in journals or conference proceedings indexed in the *Scopus* and in the *Web Of Science* databases. Please refer to Chapter 1 for a correspondence between chapters and publication.

Journal Publications

1. Ceschia, S., Di Gaspero, L., Rosati, R.M., Schaerf, A., 2022. Multi-neighborhood simulated annealing for the minimum interference frequency assignment problem. EURO Journal on Computational Optimization , 100024.

2. Rosati, R.M., Petris, M., Di Gaspero, L., Schaerf, A., 2022. Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. Journal of Scheduling 25, 301–319.

3. Rosati RM., Bouamama S., Blum C., 2024, Multi-constructor CMSA for the maximum disjoint dominating sets problem. Computers & Operations Research 161:106450

Conference proceedings

5. Rosati, R.M., Bouamama, S., Blum, C., 2023. Construct, merge, solve and adapt applied to the maximum disjoint dominating sets problem, in: Metaheuristics: 14th International Conference, MIC 2022, Syracuse, Italy, July 11–14, 2022, Proceedings, Springer. pp. 306–321.

6. Rosati, R.M., Kletzander, L., Blum, C., Musliu, N., Schaerf, A., 2023. Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. AIxIA 2022 – Advances in Artificial Intelligence. AIxIA 2022. Lecture Notes in Computer Science, vol 13796, pp 254–267

7. Ceschia, S., Di Gaspero, L., Rosati, R.M., Schaerf, A. (2024). Reinforcement Learning for Multi-Neighborhood Local Search in Combinatorial Optimization. Machine Learning, Optimization, and Data Science. LOD 2023. Lecture Notes in Computer Science, vol 14506, pp 206–221.

Submitted for publication

8. Van Bulck, D., Goossens, D., Clarner, J., Dimitsas, A., Fonseca G.H.G., Lamas-Fernandez, C., Lester, M.M., Pedersen, J., Phillips, A.E., Rosati, R.M., Which algorithm to select in sports timetabling? [submitted].

9. Ceschia, S., Di Gaspero, L., Rosati, R.M., Schaerf, A., Multi-neighborhood simulated annealing for the home healthcare routing and scheduling problem. [submitted].

10. Rosati, R.M., Schaerf, A., Multi-neighborhood simulated annealing for the capacitated dispersion problem. [submitted].

Bibliography

Aardal K, Hurkens C, Lenstra J, Tiourine S (1996) Algorithms for frequency assignment problems. CWI Quarterly 9(1-2):1–8

Aardal K, Hurkens C, Lenstra JK, Tiourine S (2002) Algorithms for radio link frequency assignment: The calma project. Operations Research 50(6):968–980

Aardal KI, Van Hoesel SP, Koster AM, Mannino C, Sassano A (2007) Models and solution techniques for frequency assignment problems. Annals of Operations Research 153(1):79–129

Adriaensen S, Nowé A (2016) Case study: An analysis of accidental complexity in a state-of-the-art hyper-heuristic for hyflex. In: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 1485–1492

Adriaensen S, Biedenkapp A, Shala G, Awad N, Eimer T, Lindauer M, Hutter F (2022) Automated dynamic algorithm configuration. Journal of Artificial Intelligence Research 75:1633–1699

Ait Haddadene SR, Labadie N, Prodhon C (2016) A GRASP× ILS for the vehicle routing problem with time windows, synchronization and precedence constraints. Expert Systems with Applications 66:274–294

Akbay MA, López Serrano A, Blum C (2022) A self-adaptive variant of cmsa: application to the minimum positive influence dominating set problem. International Journal of Computational Intelligence Systems 15(1):44

Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E (2002) Wireless sensor networks: a survey. Computer networks 38(4):393–422

Alicastro M, Ferone D, Festa P, Fugaro S, Pastore T (2021) A reinforcement learning iterated local search for makespan minimization in additive

manufacturing machine scheduling problems. Computers & Operations Research 131:105272

Allen JD, Helgason RV, Kennington JL (1987) The frequency assignment problem: A solution via nonlinear programming. Naval Research Logistics 34:133–139

Anagnostopoulos A, Michel L, Van Hentenryck P, Vergados Y (2006) A simulated annealing approach to the traveling tournament problem. Journal of Scheduling 9(2):177–193

Anderson LG (1973) A simulation study of some dynamic channel assignment algorithms in a high capacity mobile telecommunications system. IEEE Transactions on Communications 21:1294–1301

Angelelli E, Mansini R, Speranza MG (2010) Kernel search: A general heuristic for the multi-dimensional knapsack problem. Computers & Operations Research 37(11):2017–2026

Applegate DL, Bixby RE, Chvátal V, Cook W, Espinoza DG, Goycoolea M, Helsgaun K (2009) Certification of an optimal tsp tour through 85,900 cities. Operations Research Letters 37(1):11–15

Aranha C, Camacho Villalón CL, Campelo F, Dorigo M, Ruiz R, Sevaux M, Sörensen K, Stützle T (2022) Metaphor-based metaheuristics, a call for action: the elephant in the room. Swarm Intelligence 16(1):1–6

Balbal S, Bouamama S, Blum C (2021) A greedy heuristic for maximizing the lifetime of wireless sensor networks based on disjoint weighted dominating sets. Algorithms 14(6):170

Barabási AL, Albert R (1999) Emergence of scaling in random networks. science 286(5439):509–512

Bartz-Beielstein T, et al (2020) Benchmarking in optimization: Best practice and open issues. arXiv abs/2007.03488

Bazirha M, Kadrani A, Benmansour R (2023) Stochastic home health care routing and scheduling problem with multiple synchronized services. Annals of Operations Research 320(2):573–601

Beckmann D, Killat U (1999) Frequency planning with respect to interference minimization in cellular radio networks. Tech. Rep. TD(99) 032, COST 259, Vienna, Austria

Begur SV, Miller DM, Weaver JR (1997) An integrated spatial DSS for scheduling and routing home-health-care nurses. Interfaces 27

Bellio R, Ceschia S, Di Gaspero L, Schaerf A, Urli T (2016) Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. Computers & Operations Research 65:83–92

Bellio R, Ceschia S, Di Gaspero L, Schaerf A (2021) Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. Computers & Operations Research 132:105300

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research 290(2):405–421

Bennett CH, Bernstein E, Brassard G, Vazirani U (1997) Strengths and weaknesses of quantum computing. SIAM journal on Computing 26(5):1510–1523

Bertels S, Fahle T (2006) A hybrid setup for a hybrid scenario: Combining heuristics for the home health care problem. Computers & Operations Research 33

Berthold T, Koch T, Shinano Y (2021) MILP. Try. Repeat. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling, PATAT, vol 2, pp 403–411

Birattari M, Yuan Z, Balaprakash P, Stützle T (2010) F-race and iterated F-race: An overview. In: Experimental methods for the analysis of optimization algorithms, Springer, Berlin, pp 311–336

Björklund P, Värbrand P, Yuan D (2005) Optimized planning of frequency hopping in cellular networks. Computers & Operations Research 32(1):169–186

Blum C, Ochoa G (2021) A comparative analysis of two matheuristics by means of merged local optima networks. European Journal of Operational Research 290(1):36–56

Blum C, Pinacho P, López-Ibáñez M, Lozano JA (2016) Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. Computers & Operations Research 68:75–88

Borndörfer R, Eisenblätter A, Grötschel M, Martin A (1998) Frequency assignment in cellular phone networks. Annals of Operations Research 76:73–93

Bouamama S, Blum C, Pinacho-Davidson P (2022) A population-based iterated greedy algorithm for maximizing sensor network lifetime. Sensors 22(5):1804

Bredström D, Rönnqvist M (2008) Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European journal of operational research 191(1):19–31

Bruglieri M, Cordone R (2021) Metaheuristics for the minimum gap graph partitioning problem. Computers & Operations Research 132:105301

Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S (2003) Hyper-heuristics: An emerging direction in modern search technology. Handbook of metaheuristics pp 457–474

Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society 64(12):1695–1724

Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward JR (2019) A classification of hyper-heuristic approaches: revisited. In: Handbook of metaheuristics, Springer, pp 453–477

Calvo B, Santafé Rodrigo G (2016) scmamp: Statistical comparison of multiple algorithms in multiple problems. The R Journal, Vol 8/1, Aug 2016

Camacho-Villalón CL, Dorigo M, Stützle T (2023) Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors. International Transactions in Operational Research 30(6):2945–2971

Cardei M, Du DZ (2005) Improving wireless sensor network lifetime through power aware organization. Wireless networks 11(3):333–340

Cardei M, MacCallum D, Cheng MX, Min M, Jia X, Li D, Du DZ (2002) Wireless sensor networks with energy efficient organization. Journal of Interconnection Networks 3(03n04):213–229

Carter MW, Laporte G, Lee SY (1996) Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society 74:373–383

Ceschia S, Schaerf A (2011) Local search and lower bounds for the patient admission scheduling problem. Computers & Operations Research 38(10):1452–1463, DOI 10.1016/j.cor.2011.01.007, URL `http://www.dpia.uniud.it/schaerf/biblio/Papers/CeSc12a.pdf`

Ceschia S, Di Gaspero L, Schaerf A (2011) Tabu search techniques for the heterogeneous vehicle routing problem with time windows and carrier-dependent costs. Journal of Scheduling 14:601–615

Ceschia S, Di Gaspero L, Rosati RM, Schaerf A (2022) Multi-neighborhood simulated annealing for the minimum interference frequency assignment problem. EURO Journal on Computational Optimization 10:100024

Ceschia S, Di Gaspero L, Schaerf A (2023) Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research 308(1):1–18

Ceschia S, Di Gaspero L, Rosati RM, Schaerf A (2024a) Multi-neighborhood simulated annealing for the home healthcare routing and scheduling problem

Ceschia S, Di Gaspero L, Rosati RM, Schaerf A (2024b) Reinforcement learning for multi-neighborhood local search in combinatorial optimization. In: Machine Learning, Optimization, and Data Science, Springer Nature Switzerland, Cham, pp 206–221

Chang GJ (1994) The domatic number problem. Discrete Mathematics 125(1-3):115–122

Cheng E, Rich J (1998) A home health care routing and scheduling problem. Tech. Rep. CAAM TR98–04, Rice University

Chiarandini M, Stützle T (2007) Stochastic local search algorithms for graph set T-colouring and frequency assignment. Constraints 12(3):371–403

Chuang CY (2020) Combining multiple heuristics: Studies on neighborhood-base heuristics and sampling-based heuristics. PhD thesis, Carnegie Mellon University

Cissé M, Yalçındağ S, Kergosien Y, Şahin E, Lenté C, Matta A (2017) OR problems related to home health care: A review of relevant routing and scheduling problems. Operations research for health care 13:1–22

Clapper Y, Berkhout J, Bekker R, Moeke D (2023) A model-based evolutionary algorithm for home health care scheduling. Computers & Operations Research 150:106081

Cockayne EJ, Hedetniemi ST (1975) Optimal domination in graphs. IEEE Transactions on circuits and systems 22(11):855–857

Cockayne EJ, Hedetniemi ST (1977) Towards a theory of domination in graphs. Networks 7(3):247–261

Constantino AA, de Mendonça Neto CFX, de Araujo SA, Landa-Silva D, Calvi R, dos Santos AF (2017) Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm. Journal of Universal Computer Science 23(5):479–504

Correia LM (ed) (2001) Wireless Flexible Personalized Communications - COST 259: European Co-operation in Mobile Radio Research. John Wiley & Sons, COST Action 259—Final Report

Costa D (1995) An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR 33(3):161–178

Costa FN, Urrutia S, Ribeiro CC (2012) An ILS heuristic for the traveling tournament problem with predefined venues. Annals of Operations Research 194(1):137–150

Crisan C, Mühlenbein H (1998) The frequency assignment problem: A look at the performance of evolutionary search. Lecture Notes in Computer Science 1363:263–274

Croes GA (1958) A method for solving traveling-salesman problems. Operations research 6(6):791–812

De Leone R, Festa P, Marchitto E (2011) Solving a bus driver scheduling problem with randomized multistart heuristics. International Transactions in Operational Research 18(6):707–727

Decerle J, Grunder O, El Hassani AH, Barakat O (2018) A memetic algorithm for a home health care routing and scheduling problem. Operations research for health care 16:59–71

Della Croce F, Tadei R, Asioli P (1999) Scheduling a round robin tennis tournamentunder courts and players availability constraints. Annals of Operations Research 92:349–361

Demeester P, Souffriau W, De Causmaecker P, Vanden Berghe G (2010) A hybrid tabu search algorithm for automatically assigning patients to beds. Artificial Intelligence in Medicine 48(1):61–70, DOI https://doi.org/10.1016/j.artmed.2009.09.001, URL `https://www.sciencedirect.com/science/article/pii/S0933365709001341`

Desrochers M, Soumis F (1989) A Column Generation Approach to the Urban Transit Crew Scheduling Problem. Transportation Science 23(1):1–13

Di Gaspero L, Schaerf A (2003) Multi-neighbourhood local search with application to course timetabling. In: Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science, vol 2740, pp 262–275

Di Gaspero L, Schaerf A (2006) Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modeling and Algorithms 5(1):65–89

Di Gaspero L, Schaerf A (2007) A composite-neighborhood tabu search approach to the traveling tournament problem. Journal of Heuristics 13(2):189–207

Di Gaspero L, Urli T (2014) A CP/LNS approach for multi-day homecare scheduling problems. In: Blesa MJ, Blum C, Voß S (eds) Hybrid Metaheuristics, Springer International Publishing, Cham, pp 1–15

Dimitsas A, Gogos C, Valouxis C, Tzallas A, Alefragis P (2022) A pragmatic approach for solving the sports scheduling problem. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling, PATAT, vol 3, pp 195–207

Dinitz JH, Garnick DK, McKay BD (1994) There are 526,915,620 nonisomorphic one-factorizations of k_{12}. Journal of Combinatorial Design 2:273–285

Doerr B, Doerr C (2020) Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. Theory of Evolutionary Computation: Recent Developments in Discrete Optimization pp 271–321

Dorigo M, Birattari M, Stützle T (2006) Ant colony optimization. IEEE computational intelligence magazine 1(4):28–39

Drake JH, Kheiri A, Özcan E, Burke EK (2020) Recent advances in selection hyper-heuristics. European Journal of Operational Research 285(2):405–428

Dupin N, Talbi EG (2021) Matheuristics to optimize refueling and maintenance planning of nuclear power plants. Journal of Heuristics 27(1-2):63–105

Duque-Antón M, Kunz D, Rüber B (1993) Channel assignment for cellular radio using simulated annealing. IEEE Transactions on Vehicular Technology 42:14–21

Easton K, Nemhauser G, Trick M (2001) The traveling tournament problem description and benchmarks. In: Seventh International Conference on the Principles and Practice of Constraint Programming (CP 99), Springer-Verlag, LNCS, vol 2239, pp 580–589

Eisenblätter A, Koster A (2000) Fap web - a website about frequency assignment problems. `fap.zib.de`, last modified Jan 2010

Erkut E, Neuman S (1991) Comparison of four models for dispersing facilities. INFOR: Information Systems and Operational Research 29(2):68–86

Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1):3–27

Eveborn P, Flisberg P, Rönnqvist M (2006) Laps care—an operational system for staff planning of home care. European Journal of Operational Research 171

Feige U, Halldórsson MM, Kortsarz G, Srinivasan A (2002) Approximating the domatic number. SIAM Journal on computing 32(1):172–195

Feo TA, Resende MG (1995) Greedy randomized adaptive search procedures. Journal of global optimization 6:109–133

Ferrer J, Chicano F, Ortega-Toro JA (2021) CMSA algorithm for solving the prioritized pairwise test data generation problem in software product lines. Journal of Heuristics 27:229–249

Fialho A, Da Costa L, Schoenauer M, Sebag M (2010) Analyzing bandit-based adaptive operator selection mechanisms. Annals of Mathematics and Artificial Intelligence 60(1-2):25–64

Fikar C, Hirsch P (2017) Home health care routing and scheduling: A review. Computers & Operations Research 77:86–95

Fischetti M, Lodi A (2003) Local branching. Mathematical programming 98:23–47

Flood MM (1956) The traveling-salesman problem. Operations research 4(1):61–75

Fonseca GHG, Toffolo TAM (2022) A fix-and-optimize heuristic for the ITC2021 sports timetabling problem. J Sched 25:273–286

Franzin A, Stützle T (2019) Revisiting simulated annealing: A component-based analysis. Computers & Operations Research 104:191–206

Galinier P, Hertz A (2006) A survey of local search methods for graph coloring. Computers & Operations Research 33(9):2547–2562

Garcia S, Herrera F (2008) An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of machine learning research 9(Dec):2677–2694

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA

Gelling EN (1973) On 1-factorizations of the complete graph and the relationship to round robin schedules. PhD thesis

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13(5):533–549

Glover F (1989) Tabu search. Part I. ORSA Journal of Computing 1:190–206

Glover F (1990) Tabu search. Part II. ORSA Journal of Computing 2:4–32

Glover F, Kochenberger G (2006) Handbook of metaheuristics, vol 57. Springer Science & Business Media

González-Velarde JL, Laguna M (2002) Tabu search with simple ejection chains for coloring graphs. Annals of Operations Research 117(1-4):165–174

Gopalakrishnan B, Johnson EL (2005) Airline Crew Scheduling: State-of-the-Art. Annals of Operations Research 140(1):305–337

Grenouilleau F, Legrain A, Lahrichi N, Rousseau LM (2019) A set partitioning heuristic for the home health care routing and scheduling problem. European Journal of Operational Research 275(1):295–303

Grieco L, Utley M, Crowe S (2021) Operational research applied to decisions in home health care: A systematic literature review. Journal of the Operational Research Society 72(9):1960–1991

Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp 212–219

Gunawan A, Lau HC, Lu K (2018) Adopt: Combining parameter tuning and adaptive operator ordering for solving a class of orienteering problems. Computers & Industrial Engineering 121:82–96

Hamiez JP, Hao JK (2000) Solving the sports league scheduling problem with tabu search. In: Workshop on Local Search for Planning and Scheduling, Springer, pp 24–36

Hammersley JM, Handscomb DC (1964) Monte Carlo methods. Chapman and Hall, London

Hansen P, Mladenović N, Todosijević R, Hanafi S (2017) Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization 5(3):423–454

Haynes TW, Hedetniemi S, Slater P (2013) Fundamentals of domination in graphs. CRC press

Hellebrandt M, Heller H (2000) A new heuristic method for frequency assignment. Tech. Rep. TD(00) 003, COST 259, Valencia, Spain

Helsgaun K (2000) An effective implementation of the lin–kernighan traveling salesman heuristic. European journal of operational research 126(1):106–130

Holland JH (1992) Genetic algorithms. Scientific american 267(1):66–73

Huang C, Li Y, Yao X (2020) A survey of automatic parameter tuning methods for metaheuristics. IEEE Trans on Evolutionary Computation 24(2):201–216

Hutter F, Hoos H, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization, Springer, pp 507–523

Ibarra-Rojas O, Delgado F, Giesen R, Muñoz J (2015) Planning, operation, and control of bus transport systems: A literature review. Transportation Research Part B: Methodological 77:38–75

Irving RW (1991) On approximating the minimum independent dominating set. Information Processing Letters 37(4):197–200

Islam K, Akl SG, Meijer H (2009) Maximizing the lifetime of wireless sensor networks through domatic partition. In: 2009 IEEE 34th Conference on Local Computer Networks, IEEE, pp 436–442

Januario T, Urrutia S (2016) A new neighborhood structure for round robin scheduling problems. Computers & Operations Research 70:127–139

Jin J, Crainic TG, Løkketangen A (2012) A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. European Journal of Operational Research 222(3):441–451

Johnson DS, Aragon CR, McGeoch LA, Schevon C (1989) Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. Operations Research 37(6):865–892

Kadioglu S, Malitsky Y, Sellmann M, Tierney K (2010) ISAC–instance-specific algorithm configuration. In: ECAI 2010, IOS Press, pp 751–756

Kallestad J, Hasibi R, Hemmati A, Sörensen K (2023) A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. European Journal of Operational Research 309(1):446–468

Kapsalis A, Chardaire P, Rayward-Smith VJ, Smith GD (1995) The radio link frequency assignment problem: A case study using genetic algorithms. Lecture Notes on Computer Science 993:117–131

Karimi-Mamaghan M, Mohammadi M, Meyer P, Karimi-Mamaghan AM, Talbi EG (2022) Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. European Journal of Operational Research 296(2):393–422

Karmarkar N (1984) A new polynomial-time algorithm for linear programming. In: Proceedings of the sixteenth annual ACM symposium on Theory of computing, pp 302–311

Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations, Springer, pp 85–103

Kendall G, Knust S, Ribeiro C, Urrutia S (2010) Scheduling in sports: An annotated bibliography. Computers & Operations Research 37(1):1–19

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks, IEEE, vol 4, pp 1942–1948

Khachiyan LG (1979) A polynomial algorithm in linear programming. In: Doklady Akademii Nauk, Russian Academy of Sciences, vol 244, pp 1093–1096

Kheiri A, Gretsista A, Keedwell E, Lulli G, Epitropakis MG, Burke EK (2021) A hyper-heuristic approach based upon a hidden markov model for the multi-stage nurse rostering problem. Computers & Operations Research 130:105221

Kiouche AE, Bessedik M, Benbouzid-SiTayeb F, Keddar MR (2020) An efficient hybrid multi-objective memetic algorithm for the frequency assignment problem. Engineering Applications of Artificial Intelligence 87:103265

Kirkpatrick S, Gelatt D, Vecchi M (1983) Optimization by simulated annealing. Science 220:671–680

Kletzander L, Musliu N (2020) Solving large real-life bus driver scheduling problems with complex break constraints. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol 30, pp 421–429

Kletzander L, Musliu N (2022) Hyper-heuristics for personnel scheduling domains. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol 32, pp 462–470

Kletzander L, Musliu N, Van Hentenryck P (2021) Branch and price for bus driver scheduling with complex break constraints. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 35, pp 11853–11861

Kletzander L, Mazzoli TM, Musliu N (2022) Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp 232–240

Knust S (2010) Classification of literature on sports scheduling. `http://www2.informatik.uni-osnabrueck.de/knust/sportssched/sportlit_class/`, last accessed: 20/10/2023

Kolen A (2007) A genetic algorithm for the partial binary constraint satisfaction problem: an application to a frequency assignment problem. Statistica Neerlandica 61(1):4–15

Koster AMCA, van Hoesel CPM, Kolen AWJ (1999) Optimal solutions for a frequency assignment problem via tree-decomposition. Lecture Notes in Computer Science 1665:338–349

Koster AMCA, van Hoesel SPM, Kolen AWJ (2002) Solving partial constraint satisfaction problems with tree decomposition. Networks 40(3):170–180

Kummer AF (2021) A study on the home care routing and scheduling problem. PhD thesis, Universidade Federal do Rio Grande do Sul

Kummer AF, Buriol LS, de Araújo OC (2020) A biased random key genetic algorithm applied to the VRPTW with skill requirements and synchronization constraints. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp 717–724

Kummer AF, de Araújo OCB, Buriol LS, Resende MGC (2022) A biased random-key genetic algorithm for the home health care problem. International Transactions in Operational Research

Lahsinat Y, Boughaci D, Benhamou B (2018) Breakout variable neighbourhood search for the minimum interference frequency assignment problem. Journal of Systems and Information Technology 20(4):468–488

Lai X, Hao JK (2015) Path relinking for the fixed spectrum frequency assignment problem. Expert Systems with Applications 42(10):4755–4767

Laidoui F, Bessedik M, Si-Tayeb FB, Bengherbia N, Khelil MY (2018) Nash-pareto genetic algorithm for the frequency assignment problem. Procedia Computer Science 126:282–291, knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia

Lal B, Balakrishnan A, Caldwell BM, Buenconsejo RS, Carioscia SA (2018) Global trends in space situational awareness (ssa) and space traffic management (stm). Tech. rep., Institute for Defense Analyses Washington DC

Lamas-Fernandez C, Martinez-Sykora A, Potts CN (2021) Scheduling double round-robin sports tournaments. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling, vol 2, pp 435 – 448

Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. Econometrica 28(3):497–520

Landete M, Sainz-Pardo JL (2022) The domatic partition problem in separable graphs. Mathematics 10(4):640

Lasfargeas S, Gagné C, Sioud A (2019) Solving the home health care problem with temporal precedence and synchronization. In: Bioinspired Heuristics for Optimization, Springer, pp 251–267

Lester MM (2022) Pseudo-boolean optimisation for robinx sports timetabling. J Sched 25:287–299

Lewis R, Thompson J (2011) On the application of graph colouring techniques in round-robin sports scheduling. Computers & Operations Research 38(1):190–204

Lewis R, Thiruvady D, Morgan K (2019) Finding happiness: an analysis of the maximum happy vertices problem. Computers & Operations Research 103:265–276

Li J, Kwan RS (2003) A fuzzy genetic algorithm for driver scheduling. European Journal of Operational Research 147(2):334–344, fuzzy Sets in Scheduling and Planning

Li J, Pardalos PM, Sun H, Pei J, Zhang Y (2015) Iterated local search embedded adaptive neighborhood selection approach for the multi-depot

vehicle routing problem with simultaneous deliveries and pickups. Expert Systems with Applications 42(7):3551–3561

Lin DY, Hsu CL (2016) A column generation algorithm for the bus driver scheduling problem. Journal of Advanced Transportation 50(8):1598–1615

Lin K, Wang W, Wang X, Ji W, Wan J (2015) Qoe-driven spectrum assignment for 5g wireless networks using sdr. IEEE Wireless Communications 22(6):48–55

Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling-salesman problem. Operations research 21(2):498–516

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58

Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. In: Handbook of metaheuristics, Springer, pp 320–353

Lourenço HR, Paixão JP, Portugal R (2001) Multiobjective metaheuristics for the bus driver scheduling problem. Transportation Science 35(3):331–343

Lü Z, Hao JK (2012) Adaptive neighborhood search for nurse rostering. European Journal of Operational Research 218(3):865–876

Lu Z, Martínez-Gavara A, Hao JK, Lai X (2023) Solution-based tabu search for the capacitated dispersion problem. Expert Systems with Applications 223:119856

Luna F, Blum C, Alba E, Nebro AJ (2007) ACO vs EAs for solving a real-world frequency assignment problem in GSM networks. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, Association for Computing Machinery, New York, NY, USA, GECCO '07, p 94–101

Luna F, Estébanez C, León C, Chaves-González JM, Nebro AJ, Aler R, Segura C, Vega-Rodríguez MA, Alba E, Valls JM, Miranda G, Gómez-Pulido JA (2011) Optimization algorithms for large-scale real-world instances of the frequency assignment problem. Soft Computing 15(5):975–990

Mak NH, Seah WK (2009) How long is the lifetime of a wireless sensor network? In: 2009 International Conference on Advanced Information Networking and Applications, pp 763–770

Mankowska DS, Meisel F, Bierwirth C (2014) The home health care routing and scheduling problem with interdependent services. Health Care Management Science 17(1):15–30

Mannino C, Oriolo G, Ricci F, Chandran S (2007) The stable set problem and the thinness of a graph. Operations Research Letters 35(1):1–9

Mara S, Norcahyo R, Jodiawan P, Lusiantoro L, Rifai AP (2022) A survey of adaptive large neighborhood search algorithms and applications. Computers & Operations Research 146(105903)

Maron O, Moore A (1997) The racing algorithm: Model selection for lazy learners. Artificial Intelligence Review 11(1):193–225

Martello S, Toth P (1986) A heuristic approach to the bus driver scheduling problem. European Journal of Operational Research 24(1):106–117, oR and Microcomputers Miscellaneous OR Applications

Martí R, Gallego M, Duarte A (2010) A branch and bound algorithm for the maximum diversity problem. European journal of operational research 200(1):36–44

Martí R, Martínez-Gavara A, Sánchez-Oro J (2021) The capacitated dispersion problem: An optimization model and a memetic algorithm. Memetic Computing 13:131–146

Martí R, Martínez-Gavara A, Pérez-Peló S, Sánchez-Oro J (2022) A review on discrete diversity and dispersion maximization from an or perspective. European Journal of Operational Research 299(3):795–813

Méndez-Díaz I, Zabala P (2008) A cutting plane algorithm for graph coloring. Discrete Applied Mathematics 156(2):159–179

Mesbahi M, Egerstedt M (2010) Graph theoretic methods in multiagent networks. Princeton University Press

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. The journal of chemical physics 21(6):1087–1092

Mischek F, Musliu N (2022) Reinforcement learning for cross-domain hyper-heuristics. In: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, pp 4793–4799

Misir M, De Causmaecker P, Vanden Berghe G, Verbeeck K (2011) An adaptive hyper-heuristic for chesc 2011. In: OR53 Annual Conference

Mladenović N, Hansen P (1997) Variable neighborhood search. Computers & Operations Research 24(11):1097–1100

Mladenović N, Todosijević R, Urošević D, Ratli M (2022) Solving the capacitated dispersion problem with variable neighborhood search approaches: From basic to skewed vns. Computers & Operations Research 139:105622

Montemanni R (2001) Upper and lower bounds for the fixed spectrum frequency assignment problem. PhD thesis, University of Glamorgan

Montemanni R, Smith DH (2010) Heuristic manipulation, tabu search and frequency assignment. Computers & Operations Research 37(3):543–551

Montemanni R, Smith D, Allen SM (2001) Lower bounds for fixed spectrum frequency assignment. Annals of Operations Research 107(1-4):237–250

Montemanni R, Moon JN, Smith DH (2003) An improved tabu search algorithm for the fixed-spectrum frequency-assignment problem. IEEE transactions on vehicular technology 52(4):891–901

Montemanni R, Smith D, Allen S (2004) An improved algorithm to determine lower bounds for the fixed spectrum frequency assignment problem. European Journal of Operational Research 156(3):736–751

Mosadegh H, Ghomi S, Süer G (2020) Stochastic mixed-model assembly line sequencing problem: Mathematical modeling and Q-learning based simulated annealing hyper-heuristics. European Journal of Operational Research 282(2):530–544

Murphey RA, Pardalos PM, Resende MG (1999) Frequency assignment problems. Handbook of Combinatorial Optimization: Supplement Volume A pp 295–377

Nguyen TN, Huynh DT (2007) Extending sensor networks lifetime through energy efficient organization. In: International Conference on Wireless Algorithms, Systems and Applications (WASA 2007), IEEE, pp 205–212

Oladzad-Abbasabady N, Tavakkoli-Moghaddam R, Mohammadi M, Vahedi-Nouri B (2023) A bi-objective home care routing and scheduling problem considering patient preference and soft temporal dependency constraints. Engineering Applications of Artificial Intelligence 119:105829

Ore O (1962) Theory of graphs

Peiró J, Jiménez I, Laguardia J, Martí R (2021) Heuristics for the capacitated dispersion problem. International transactions in operational research 28(1):119–141

Phillips AE, O'Sullivan M, Walker C (2021) An adaptive large neighbourhood search matheuristic for the ITC2021 sports timetabling competition. In: Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling, PATAT, vol 2, pp 426 – 430

Pinacho-Davidson P, Bouamama S, Blum C (2019) Application of CMSA to the minimum capacitated dominating set problem. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp 321–328

Pisinger D, Ropke S (2019) Large neighborhood search. In: Handbook of metaheuristics, Springer, pp 99–127

Poon SH, Yen WCK, Ung CT (2012) Domatic partition on several classes of graphs. In: International Conference on Combinatorial Optimization and Applications, Springer, pp 245–256

Portugal R, Lourenço HR, Paixão JP (2008) Driver scheduling problem modelling. Public Transport 1(2):103–120

Preskill J (2023) Quantum computing 40 years later. In: Feynman Lectures on Computation, CRC Press, pp 193–244

Rasmussen MS, Justesen T, Dohn A, Larsen J (2012) The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. European Journal of Operational Research 219

Rasmussen RV, Trick MA (2008) Round robin scheduling–a survey. European Journal of Operational Research 188(3):617–636

Rendl A, Prandtstetter M, Hiermann G, Puchinger J, Raidl G (2012) Hybrid heuristics for multimodal homecare scheduling. In: Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, Springer, Heidelberg, pp 339–355

Ribeiro CC, Urrutia S (2007) Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research 179(3):775–787

Riege T, Rothe J (2005) An exact 2.9416^n algorithm for the three domatic number problem. In: International Symposium on Mathematical Foundations of Computer Science, Springer, pp 733–744

Riege T, Rothe J, Spakowski H, Yamamoto M (2007) An improved exact algorithm for the domatic number problem. Information Processing Letters 101(3):101–106

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40(4):455–472

Rosa A, Wallis WD (1982) Premature sets of 1-factors or how not to schedule round robin tournaments. Discrete Applied Mathematics 4:291–297

Rosati RM, Schaerf A (2024) Multi-neighborhood simulated annealing for the capacitated dispersion problem

Rosati RM, Petris M, Di Gaspero L, Schaerf A (2022) Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. Journal of Scheduling 25(3):301–319

Rosati RM, Bouamama S, Blum C (2023a) Construct, merge, solve and adapt applied to the maximum disjoint dominating sets problem. In: Metaheuristics, pp 306–321

Rosati RM, Kletzander L, Blum C, Musliu N, Schaerf A (2023b) Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. In: AIxIA 2022 – Advances in Artificial Intelligence, pp 254–267

Rosati RM, Bouamama S, Blum C (2024) Multi-constructor CMSA for the maximum disjoint dominating sets problem. Computers & Operations Research 161:106450

Rosenkrantz DJ, Tayi GK, Ravi S (2000) Facility dispersion problems under capacity and cost constraints. Journal of Combinatorial Optimization 4:7–33

Russell KG (1980) Balancing carry-over effects in round robin tournaments. Biometrika 67(1):127–131

dos Santos JPQ, de Melo JD, Neto ADD, Aloise D (2014) Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search. Expert Systems with Applications 41(10):4939–4949

Schneider M, Hoos HH (2012) Quantifying homogeneity of instance sets for algorithm configuration. In: International Conference on Learning and Intelligent Optimization, Springer, pp 190–204

Segura C, Hernández-Aguirre A, Luna F, Alba E (2016) Improving diversity in evolutionary algorithms: New best solutions for frequency assignment. IEEE Transactions on Evolutionary Computation 21(4):539–553

Shahmardan A, Sajadieh M (2020) Truck scheduling in a multi-door cross-docking center with partial unloading–reinforcement learning-based simulated annealing approaches. Computers & Industrial Engineering 139:106134

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: Proc. of the 4th Int. Conf. on Principles and Practice of Constraint Programming (CP-98), Lecture Notes in Computer Science, vol 1520, Springer-Verlag, pp 417–431

Shen Y, Kwan RSK (2001) Tabu Search for Driver Scheduling. In: Computer-Aided Scheduling of Public Transport, vol 505, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 121–135, series Title: Lecture Notes in Economics and Mathematical Systems

Shier DR (1977) A min-max theorem for p-center problems on a tree. Transportation Science 11(3):243–252

Siddiqi UF, Sait SM (2018) An optimization heuristic based on non-dominated sorting and tabu search for the fixed spectrum frequency assignment problem. IEEE Access 6:72635–72648

Smith BM, Wren A (1988) A bus crew scheduling system using a set covering formulation. Transportation Research Part A: General 22(2):97–108

Smith-Miles K, Muñoz MA (2023) Instance space analysis for algorithm testing: Methodology and software tools. ACM Computing Surveys 55(12):1–31

Song H, Triguero I, Özcan E (2019) A review on the self and dual interactions between machine learning and optimisation. Progress in Artificial Intelligence 8(2):143–165

Sörensen K (2015) Metaheuristics—the metaphor exposed. International Transactions in Operational Research 22(1):3–18

Soto M, Sevaux M, Rossi A, Reinholz A (2017) Multiple neighborhood search, tabu search and ejection chains for the multi-depot open vehicle routing problem. Computers & Industrial Engineering 107:211–222, DOI https://doi.org/10.1016/j.cie.2017.03.022, URL `https://www.sciencedirect.com/science/article/pii/S0360835217301183`

Sutton R, Barto A (2018) Reinforcement learning: An introduction. MIT press

Talbi EG (2021) Machine learning into metaheuristics: A survey and taxonomy. ACM Computing Surveys (CSUR) 54(6):1–32

Tiourine SR, Hurkens CAJ, Lenstra JK (2000) Local search algorithms for the radio link frequency assignment problem. Telecommunication systems 13(2):293–314

Toffolo TA, Christiaens J, Van Malderen S, Wauters T, Vanden Berghe G (2018) Stochastic local search with learning automaton for the swap-body vehicle routing problem. Computers & Operations Research 89:68–81

Urli T (2013) json2run: a tool for experiment design & analysis. Computers & Operations Research abs/1305.1112

Van Bulck D, Goossens D (2023a) First-break-heuristically-schedule: Constructing highly-constrained sports timetables. Operations Research Letters 51:326–331

Van Bulck D, Goossens D (2023b) The international timetabling competition on sports timetabling (ITC2021). European Journal of Operational Research 308(3):1249–1267

Van Bulck D, Goossens D, Schönberger J, Guajardo M (2020) Robinx: A three-field classification and unified data format for round-robin sports timetabling. European Journal of Operational Research 280(2):568–580

Van Bulck D, Goossens D, Clarner JP, Dimitsas A, Fonseca GHG, Lamas-Fernandez C, Lester MM, Pedersen J, Phillips AE, Rosati RM (2023) Which algorithm to select in sports timetabling? `2309.03229`

Van-Rooij JMM (2010) Polynomial space algorithms for counting dominating sets and the domatic number. In: International Conference on Algorithms and Complexity, Springer, pp 73–84

Černý V (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications 45(l):41–51

Wallis WD (1983) A tournament problem. Journal of the Australian Mathematical Society Series B 24:289–291

Wang P, Henz B (2017) Frequency assignment for joint aerial layer network high-capacity backbone. Tech. rep., Army Research Laboratory Aberdeen Proving Ground United States

Watson JP (2010) An Introduction to Fitness Landscape Analysis and Cost Models for Local Search, Springer US, Boston, MA, pp 599–623

Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world'networks. nature 393(6684):440–442

Welsh DJ, Powell MB (1967) An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal 10(1):85–86

de Werra D (1981) Scheduling in sports. In: Hansen P (ed) Studies on Graphs and Discrete Programming, North Holland, pp 381–395

de Werra D, Jacot-Descombes L, Masson P (1990) A constrained sports scheduling problem. Discrete Applied Mathematics 26:41–49

Weyland D (2015) A critical analysis of the harmony search algorithm—how not to solve sudoku. Operations Research Perspectives 2:97–105

Wren A (2004) Scheduling vehicles and their drivers-forty years' experience. Tech. rep., University of Leed

Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Annals of Operations Research 196:611–634

Xiang T, Li Y, Szeto WY (2021) The daily routing and scheduling problem of home health care: based on costs and participants' preference satisfaction. International Transactions in Operational Research

Zhang D, Piao M, Zhang T, Chen C, Zhu H (2020) New algorithm of multi-strategy channel allocation for edge computing. AEU-International Journal of Electronics and Communications 126:153372

Zhao L, Wang H, Zhong X (2018) Interference graph based channel assignment algorithm for D2D cellular networks. IEEE Access 6:3270–3279